AUSTRALIAN UNIX USERS GROUP NEWSLETTER

To Print or Not To Print

Here is yet another bumper issue of AUUGN. Yes, I know its a bit late, but I have been having my house renovated, and that comes before this.

Due to financial limitations I have had to adopt some fairly strict control over what has gone into this issue and reserve a lot of material for inclusion in latter issues. The editorial committee (an ad hoc group of people including John Lions and Kevin Hill) have been asked to comment on what should be included, but unfortunately considerations such as postal weight have forced a number of items into the list below. This list gives a brief summary of the stuff I have on hand, available on request, which may be included in future AUUGNs.

Various advertisements from the last US meeting offering UNIX or UNIX-like software and/or documentation.

"Programming in QED: A Tutorial" plus a Qed manual entry

Writeup on the Georgia Tech 'Software Tools' Subsystem

An invitation to a US general access UNIX network

Quite a large manual entry for the Tilbrook Information System

A list of names and affiliations of the 450 or so attendees at the US meeting

"Macros for Analyzing C Program Arguments" by John Lions

A transcription from a tape that Ian Johnstone sent us about the US software tools and UNIX meetings (suitably edited).

A study of the way UNIX boots itself into operation.

Most of the normal correspondence.

You may also notice a slight change in the page length of this issue. People, particularly in the USA, have pointed out that it is very difficult to copy Australian 'A4' pages on to smaller pages used as standard in other countries. All contributors please note the need for more white space.

US User Meeting Summary

Included in this issue is a rather detailed review of the last US meeting.

Overseas Arrangements

This issue contains material from the last UKUUGN which I received recently as part of the exchange agreement with the UK users group. I have written, formally requesting similar agreements to the two US users groups plus the smaller (no newsletter etc.) Canadian group. I hope to have exchange arrangements agreed upon by late May and obtain material for the next newsletter.

Still More About Vaxes

A VAX11/780 will be delivered to the School of Electrical Engineering at the University of New South Wales late in June 1980. It will be used for teaching in the Department of Computer Science, to relieve the massive overloading experienced by the Department over the last year or more.

For those of you interested in the Berkeley VAX software, this newsletter has four papers on design, implementation, experience and evaluation of this software. Should these papers stir you to purchase VMUNIX, I have also enclosed a license agreement.

International Meeting

The International UNIX meeting has been planned for 18th / 19th of October. Robert Elz has written detailing progress and imploring users, both local and overseas to reply to the announcement even if they are not comming or do not know for certain. His letter appears latter in this issue. Considering that eight of the fifteen replies so far received have been from the UK and the USA, local group members are showing disproportionate apathy.

Dial-in Facilities

There are a few dial-in lines available to our readers at the Uni of New South Wales, and Sydney Uni. The Australian Graduate School of Management, where the local software data-base is maintained allows visitors to login on (02)662-1666 for read-only access to software. Mail 'kev' on that machine if you cant find something.

The Elec Eng machine allows access on (02)662-1733. Mail to the AUUGN editor may be sent to 'auugn' and some other noteable persons have accounts on this machine including John Lions ('johnl').

The Basser Dept of Computer Science at Sydney Uni, on (02)660-5772, is the entry to a small subsection of the local network. Mailing on this machine it is possible to contact:

piers	Piers Dick-Lauder
chris	Chris Maltby
chrisr	Chris Rowles (large mini UNIX person)

The login name for all the above numbers is 'visitor'.

The Software Catalogue

To date I have received a grand total of four (4) replies, pledging between six and seven hundred dollars towards the project. At the last meeting I counted at least a dozen people who said their site would be prepared to contribute a substantial amount of money.

I am amazed by the apathy of you readers out there. A site in ISRAEL who had not even subscribed until this issue took the trouble to obtain a software catalogue form and offer to make a donation. Yet all you locals could only manage three replies!!!!!

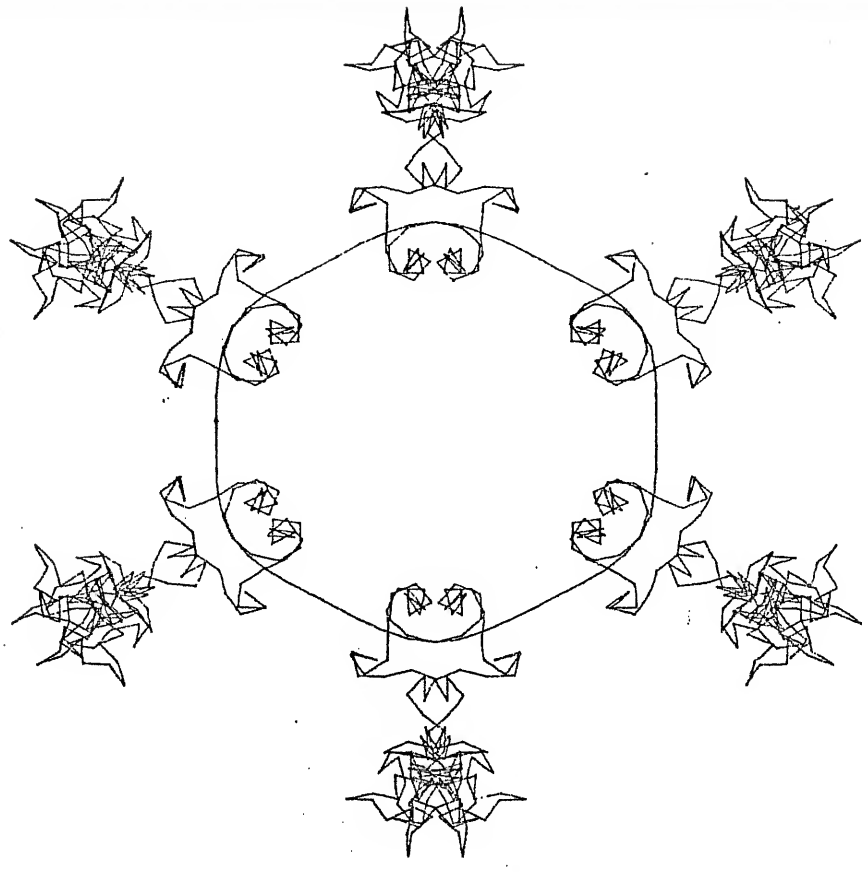
I know that students work for peanuts, but at least chip in to by the paper bag to put the food in!!!

Site Surveys

Well considering there are only four sites in our whole readership I wonder how the rest of you do any computing. Only four replies, not the same four as above, from a total readership of sixty-five. Honestly, I feel that if I enclosed a postage payed envelope to return forms in, you still would not fill them in and post them.

I have included a site survey form from the UK group in the hope that a second (non-manufacturer-biased) attempt may get a better response. Unless some more enthusiasm is shown I shall not even bother to send forms out (be they anti Perkin-Elmer or not - thanks for your reply Juris).

6-handed Bridge?



Peter Ivanov
Dept. of Computer Science
Electrical Engineering
PO Box 1
Kensington 2033
AUSTRALIA

(02) 662-3781

TELEPHONE
345 1844

TELEGRAMS
UNIMELB PARKVILLE

University of Melbourne

DEPARTMENT OF COMPUTER SCIENCE

Parkville, Victoria 3052

12th April, 1980

Peter Ivanov,
Dept. of Computer Science,
Electrical Engineering,
University of New South Wales,
Kensington.

Dear Peter,

As I mentioned on the 'phone the other day, the International Unix User's Group meeting planned for Melbourne later this year will be held on the weekend of the 18th / 19th of October. These dates were the most preferred of those that were proposed, in fact there has been no request for any of the other dates at all.

Times, venue, and program will be announced later, when I have a better idea of the number of attendees. As yet there have been very few replies to the request for people to let me know whether they intend coming or not. I would appreciate it if you could request the readers of AUUGN, (both local and overseas) who have not yet so indicated, to do so as soon as possible. Please emphasise that I would prefer a note indicating that the sender is not coming, or is uncertain to no reply at all, as I will then be able to form a more accurate estimate of the size of the meeting.

In order that you may be able to induce some replies, I am enclosing a list of those who have replied, others may be shamed into doing likewise.

The availability of accomodation at reasonable prices for interstate and overseas guests is currently being investigated. The prospects in this area have brightened slightly since our earlier conversation, I will send more details when available. Anyone who would be interested in

such accomodation (either just for the UUG meeting, or stretching backwards into the previous week for IFIP) should NOT wait for such further information to come via the newsletter, but should contact me as soon as possible.

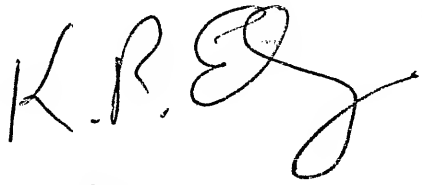
On an entirely unrelated matter, prehaps either you or one (or more) of the readers of AUUGN could provide a solution to the following problem, which is probably a UNIX bug, or if not, I would like to know where I have missed something.

If an 'open' or 'close' routine of a device driver sleeps at low priority (> PZERO), and gets interrupted by a signal to the calling process, strange things happen. If an open routine is interrupted, UNIX regards the device as open, but returns error (EINTR) status to the process, which must assume that the open failed. (A C program will find it difficult to obtain the file descriptor in any case). If a close routine is interrupted, UNIX regards the device as closed, depriving the driver of the chance to clean up correctly (the process will almost certainly ignore the error indication here, and will regard the file as closed). In general no insurmountable problems occur, but the combination of a single use device (like a line printer) and a process that tries to be intelligent about its use of that device (ie: doesn't just give up and exit when an open fails) and which uses signals to receive messages from other processes can cause difficulties.

If anyone has a solution to this, or can suggest a suitable way to avoid the problem (especially allowing the close routine to tidy up and release resources) I would be grateful if they would let me know.

Thanks for your help in publicising the International Unix Users Group meeting,

Yours



Robert Elz.

Replies to the request for information as to who will be attending the International Unix Users Group meeting in Melbourne in October have been received from:

Piers Lauder,	Comp Sci, Sydney Uni	coming
I. R. Perry,	L.E.R.S.-Synthelabo, Paris, France	coming
Roger Miller,	Architecture, UNSW	coming
J. Reinfelds,	Comp Sci, Wollongong Uni	coming
Geoff Cole,	Computing Centre, Sydney Uni	coming
Ken Yap,	Sydney Uni	coming
A. L. Arms,	Western Electric Co Inc	coming
David C Hunt	Mathematics, UNSW	coming
G. W. Gerrity	Military Studies, UNSW	coming
T. A. Dolotta	Bell Labs	possible
E. Foxley	Mathematics, Uni Nottingham	not coming
R. J. Wolff	Inst Astronomy, Uni Hawaii	not coming
P. A. Lee	Computing Lab, Uni Newcastle upon Tyne	not coming
Colin Taylor	Westfield College, Uni London	not coming
R. P. A. Collinson	Uni Kent	not coming

I eagerly await more replies,

Robert Elz.

The notes you have received are a summary of the Software Tools and UNIX meetings in Boulder, Jan 80. Please use them as appropriate in your newsletters. Copies have been sent to those persons below, which means that individual subscribers will receive multiple copies. So be it. Better that than none at all.

Martin Tuori, Greg Hill, Ian Johnstone, David Tilbrook

Bruce Anderson
UK UNIX SIG Newsletter Editor
Electrical Engineering Science
University of Essex
COLCHESTER CO4 3SQ
England

/2 copies (including one for the Australian newsletter) to:
Ian Johnstone
Bell Labs
600 Mountain Ave. Peter I.
Murray Hill
New Jersey 07974
USA

Newsletter Editor
Usenix Association
Box 8,
The Rockefeller University
1230 York Ave.
New York, NY 10021
USA

Neil Groundwater
Software Tools Newsletter Editor
Analytic Disciplines Inc.
8320 Old Courthouse Road
Vienna, VA 22180
USA

Newsletter Editor
Login West
PO Box 584
Menlo Park
California 94025
USA

Mike Tilson
Canadian UNIX Sig Newsletter Editor
Human Computing Resources Corp.
40 St. Mary Street
Toronto, Ontario
M4Y 1P9

THE BOULDER SOFTWARE TOOLS AND USENIX MEETINGS IN SUMMARY

SOFTWARE TOOLS and USENIX Meetings
Boulder, Colorado
January 28 -- February 2, 1980

This report is a summary of two winter meetings held in Boulder.
It is based on notes and memories of the attendees listed below, and as such reflects our personal biases and knowledge. Extensive detail has been deliberately avoided, in the hope of keeping these notes down to a reasonable size.

In general, there is a rapid growth in the size of both these users groups; there were 450 attendees at the USENIX meeting, which is the largest attendance yet. For the Software tools group this is apparent in sheer numbers, as well as in the formation of SIG's to deal with issues in specific subject matter areas and geographic regions. There is a strong cooperative attitude within this community, which will soon result in two new software distributions, and an informal UNIX network for mail and news. We have also noted a trend towards more formal reporting of technical information. The next USENIX meeting will publish a proceedings, and individuals are encouraged to send in short articles for publication in the newsletters. It is important that reviews of bug fixes, performance considerations, and available software be produced periodically, to help fill the growing gap between experienced hacks and new users.

We cannot guarantee that what is reported here was actually said. If you want to be SURE, or need more information, check with the speaker in question. Our apologies to anyone who has been misquoted.

Our thanks to the many persons who made informative presentations at the meetings. Further thanks to David Sherman, whose notes and macros from last June's conference made easy the production of these notes.

February 5, 1980

Martin Tuori
Defence and Civil Institute of Environmental Medicine
(DCIEM)
Downsview, Canada M3M 3B9
(416) 633-4240 Ext 204

Gregory Hill
University of Toronto Computer Services, SF105
University of Toronto
Toronto, Canada M5S 1A1
(416) 978-8853

Ian Johnstone formerly University of New South Wales
Bell Laboratories
600 Mountain Ave
Rm 7E-306
Murray Hill NJ 07974
(201) 582-5767

Table of Contents

SOFTWARE TOOLS USERS GROUP MEETING

TUESDAY MORNING

Chair: Debbie Scherrer, Lawrence Berkeley Laboratory

Speaker 1 9:00 a.m. **Opening Remarks**

Debbie Scherrer
Lawrence Berkeley Laboratory

Debbie introduced this meeting by giving a very brief history of the Software Tools development. The book 'Software Tools', by Kernighan and Plauger sparked the group; at the same time Addison and Wesley made available a tape containing the source for the tools described in the book, for the princely sum of $25. So far, about 800 tapes have been sold. This is the second meeting of the Software Tools Users Group.

Speaker 2 9:05 a.m. **Enhanced Tools**

Allen Akin
Georgia Institute of Technology

Allen described their ring network of 4 PRIME computers, running the Primos IV O.S. This configuration has been up for about 3 years, and has a user community of about 150, from secretaries to researchers. They have modified many of the tools, including a shell which supports multiple inputs and outputs, for network applications. They have written STACC (STill Another Compiler-Compiler), and with it have rewritten Ratfor, to allow extensions including recursion in internal procedures. They have been very active and have created many new and useful tools. Their package is available to PRIME users for $3000 for the first year, $1000 a year thereafter (schools less 30%). However, some of their tools are free, and have been submitted to the users group. At this time PRIME UK and PRIME Australia are distributing this package.

Speaker 3 9:30 a.m. **Heterogeneous Networking**

Joe Sventek
Lawrence Berkeley Laboratory

The problem with which LBL has been faced is to connect different machines in such a way that users could work across the network. This should include not just mail, virtual terminal access, and file transfers, but full resource sharing. User support is required at two levels: at the command level, standard utilities should be available throughout the network; at the programming level, a single set of system calls and primitives within Ratfor should be available throughout the network. This creates a virtual operating system which can be overlaid on any host in a network, in which system services (utilities and system calls) are consistent. So far this has been implemented under VMS, RSX, UNIX and TENEX.

A complete set of primitives might include:
File I/O: open, close, create, getline, putline, getchar, putchar,
prompt, remove, amove, rawmode.
Directory control: chdir, print—working—dir, opendir, closedir,
get—dir—parms, mkdir, rmdir, mvdir.
Processes: spawn, kill, suspend, resume, pstat, pwait.

Speaker 4 10:00 a.m. **Standardized Primitives**

Skip Egdorf
Consultant to
U.S. Geological Survey
National Earthquake Information Service

Skip has been using Ratfor to produce applications packages for the study and monitoring of earthquakes. Since the projects have moved to a new machine about once every two years, a strong portable base was needed. Ratfor was chosen for just that reason; but it needs stronged portablility/standardization in its system primitives. He suggested the following set:

I/O: getchar, putchar, getline, putline, readf, writef,
readb(binary), writeb, reads(string), writes.
File System: open, close, create, remove, seek, mark,
mkdir, rmdir.
Processes: getarg, spawn, suspend.
String Manipulation: pack, unpack.

Skip asked for comment and discussion within the users group, so that a single set of extended primitives might be developed.

---- *BREAK* ----

Speaker 5 10:45 a.m. **Naval Ocean Systems Center**

Bob Calland
Naval Ocean Systems Center

Bob described their primary use of the tools, the development of large single stand-alone programs for on-board mini's. To do this, they have developed the necessary cross-compiler, assembler, loader, etc. Their targets include a military micro, the AN/UYK-20, and the CMS2.

Speaker 6 11:15 a.m. **Portable Crystallography Software**

Jim Stewart
University of Maryland

This was one of the more entertaining talks. Jim is a chemist, whose interest in software is summarized by 'Kernighan's 3rd Principle': 'Let someone else do the hard part.' He and Robert Munn

have developed applications in crystallography, based on an improved Ratfor with a macro preprocessor -- RATMAC. Jim outlined his own evolution from hand calculation, through unit record equipment, Fortran 2, Fortran 4, to Ratfor. He sees Fortran 77 as another opportunity to rewrite all existing software -- an opportunity which he would like to decline.

Their efforts have enabled them to port their software to a large number of different systems; it is available for $100 from:
Computer Science Center
University of Maryland
College Park, Maryland 20742
attn: Dr. R. Munn
or send a letter to receive a copy of the RATMAC primer.

Speaker 7 11:45 a.m. **Maintaining TELCO Software**

Dick McLaughlin
Bell Labs

Dick described EPLANS (Engineering, Planning and Analysis Software) - a collection of programs which are used to handle the implementation and support of telephone switching networks. This represents some 30 programs, or 50,000 lines of Fortran code, which is being converted to Ratfor. This software is in use by Western Electric, Bell Canada, and independent Telephone Companies. Their experience has shown that an applications programmer faced with understanding and supporting a 9000 line Fortran program could spend 18 months, and not be able to modify the program without assistance from the authors; on the other hand, the same programmer could be up to speed within a month, working on the same program in Ratfor. Their target systems include MVS/TSO, VM/CMS, DEC System10, GCOS, UNIVAC, XEROX, and UNIX-VAX. In the future, they are looking toward EFL (Extended Fortran Language), which will allow extended data and control structures, a full compiler (not just the preprocessor), and in general a superset of Ratfor.

---- *LUNCH* ----

Speaker 8 1:30 p.m. **S -- Stats System**

Rick Becker
Bell Labs

Rick described the 'S' statistical analysis system, which he says is easy to use, powerful, extensible, and portable. He emphasized that it allows a statistician to 'come in contact' with his data, through an interpretive expression language, like APL, but using functions rather than operators:
e.g. z <-- regress(x,y)
S was developed using Ratfor, M4, YACC, STRUCT, and an interface language. (STRUCT takes Fortran and tries to turn it into Ratfor).

Rick went on to indicate the problems involved in moving S to another system: the supporting tools would need to be port'ed, there would be OS dependencies to consider, there might be conflicts with other tools, and programmers would be required to adapt. S as currently implemented requires that all its database be in main memory -- this is a problem on the 11/70, but not on the VAX. Some consideration is being given to allowing analyses to operate sequentially, where appropriate. It works

well at this time on small to medium data sets.

S is based on the GRZ Graphics Package, which is not yet available outside the Bell System. It is, however, their intention to release S at some time.

Speaker 9 2:00 p.m. **ALDS -- Statistics Package**

Jan Lewis
Battelle (Pacific Northwest Labs)

Jan described the research efforts going on at Battelle; they have a computer lab environment for testing and implementing ideas on the handling of large data sets. The work will be carried out on a VAX, with RAMTEK colour displays; the general requirements are that the ALDS system be data-directed, iterative (action at a given step depends on the results of prior phases of analysis), interactive, and graphics-intensive. They need to get up and running soon, so they expect to develop an operational system which may be discarded later. They would prefer to develop something which can be extended and modified.

The system will be extended to include Database research, using a standard file format called a self-descriptive binary, or SDB file. It is intended that ALDS will form an 'operating system' in its own right, with three main components: statistics, graphics, and database. To this end, work will be carried out to evaluate the relative merits of command versus menu-driven user interfaces. User interface tools will be developed, so that the style of user interface can be changed, without major change to the software application packages themselves.

This software will likely all be in the public domain, although specific target date has been chosen for completion.

Speaker 10 2:20 p.m. **Publication of Algorithms**

Webb Miller
ACM Algorithms Editor
University of California, Santa Barbara

Webb reviewed some of the history of algorithm development and reporting (1960-1980), and indicated several trends. Where Algol used to be the preferred vehicle for algorithm publishing, Fortran is now much more prevalent. The ratio of numerical to combinatorial algorithms has varied, but remains on the order of five/ten to one. The length of algorithms published has increased, so that they are now commonly in excess of 1000 lines of source.

Earlier work in the programming field has led to the standardization and development of useful software packages, including EISPACK (eigenvector software), NAG (numerical algorithms group), and IMSL (International Math and Stats Lib). Webb then suggested that these same program development philosophies are applicable to non-numeric software, and that Software Tools users SHOULD publish their algorithms, both for the sake of the community, and for their own accreditation. Suitable journals include Communications of the ACM, Transactions on Mathematical Software, and TOPLAS.

He went on to describe TOOLPACK, a prototype environment for the development, testing, analysis, and verification of mathematical software. It includes a Fortran-intelligent editor, but still needs a structurer to convert from C into Ratfor, EFL, SFtran3, Fortran 77 (any or all). For a "TOOLPACK

Prospectus", write:

Dr. Wayne Cowell
Applied Mathematics Division
Bldg. 221
Argonne National Labs
Argonne, Illinois 60439
(312) 972-7164

---- BREAK ----

Speaker 11 3:00 p.m. **NEW Basic Software Tools Tape**

Debbie Scherrer
Lawrence Berkeley Laboratory

Debbie indicated that they have been involved in a teleconference group, trying to establish a new standard collection of tools. There has been strong participation from LBL, Georgia Inst. of Technology, University of Arizona, and others. The result should be a three part tape, to be distributed by the U.of A. sometime in the next 6 months. For a piece of software to be included, it must satisfy the following:

- acceptable to the users group and/or the teleconference group
- adequately documented
- should be written using the basic primitives, or use well-documented local primitives
- be runnable under the Addison-Wesley Ratfor package
- be runnable at GIT, LBL, and UofA with few/no changes.
- (other volunteer sites are welcome)

PART1: the contents of the Addison-Wesley tape, and other portable, generally useful tools, including Ratfor and Macro preprocessors, editors, text handlers.
PART2: additional tools, including alternate versions of tools in part 1, a shell command interpreter, mail, screen editors, tape archiver, virtual Libraries of primitives will be included to provide support for as many Computer+Operating System combinations as possible, maybe 30 or 40.
PART 3: Member lists.

In the future, good tools will be added to the tape, and moved up from PART2 to PART1. We'll be looking for Graphics, SCCS, make, and other goodies.

Speaker 12 3:30 p.m. **Special Interest Groups**

Four informal SIGS have been formed; persons interested were invited to split off into separate areas to discuss plans and organizational details. Brief reports were solicited from these groups, as follows:

Text Processing SIG
All text tools will be included on the tape, the best in PART1, the rest in PART2.

Networking SIG
A questionaire will be distributed to members, asking for info on topologies and problems. The goal of

the SIG is a common command language, consistent across heterogeneous networks.

Primitives SIG
This SIG will attempt to layer the present superset of primitives which have been suggested. It is acknowledged that selecting a clean, yet complete set is as yet a black art, about which we need to learn much more.

RATFOR SIG
A committee was formed to select a chairperson, and evaluate the 3 Ratfor Preprocessors which have been offered. One will be selected for PART1 of the new tools tape; the others will presumably be included in PART2.

SOFTWARE TOOLS USERS GROUP NEWSLETTER
Neil Groundwater volunteered to handle the newsletter for the time being.

Neil Groundwater
Analytic Disciplines Inc.
8320 Old Courthouse Road
Vienna, VA 22180
(703) 893-6140

USENIX MEETING
WEDNESDAY JANUARY 30 MORNING SESSION
Chairperson -- Mike O'Dell

Speaker 1 **Opening Comments**

John Donnelly
NCAR

John introduced a new device, '/dev/snow', and welcomed us to the Boulder USENIX meeting. The conference was held in a theatre near the Harvest Hilton. This was necessary since 450 attended. There was a question as to whether it was merely coincidence that 'Black Hole' was showing at the time.

Speaker 2 **Converting to UNIX**

Brad Cox
Hendrix Electronics

Hendrix produces text-handling equipment; the Chicago Tribune is one of their large customers -- their shop contains 6 PDP-10's, 30 PDP-11/34's, and hundreds of TI 9900's based intelligent terminals. Brad outlined some of the problems and internal conflicts which arose in converting their central software development facility from RSTS to UNIX, assembler to C and Pascal. This conflict sounded typical of so many other conversions. In answer to a question concerning the possibility of general guidelines for such a conversion, Brad suggested that system support people work to set clear and powerful exapmles of UNIX's power and productivity.

Speaker 3 **New Products from Yourdon**

Mark Pearson
Yourdon Inc.

Mark mentioned four product areas from Yourdon:

Yourdon has a C compiler for RSX-11M, VAX native mode, and IBM systems. The price is $2500 1st cpu, $500 next -- for source; future versions may be released for the DEC 10, and the Western Digital Pascal machine.

OMNIX is a Z80-based system which supports pipes, I/O redirection, a shell, but no C compiler yet. It is NOT a UNIX, but is similar. It is multi-user, and supports a UNIX filesystem. It can run CP/M binaries, as it supports the CP/M system calls. It handles a range of peripherals, and sells for $350.

Yourdon now offers a supported binary UNIX/V6 licence, single user on an 11/23 with RL01's, for $2500. They have one of the new WE licenses which relates the royalties they pay to the size of the systems they licence.

They also have a queued spooling system for UNIX's, $1000.

---- *BREAK* ----

Speaker 4 **News from Intermetrics**

Morris Kranc
Intermetrics

Intermetrics is a software house; they use PWB/UNIX. TSO has effectively been replaced by using PWB (ed, sccs, make) + RJE. They have encountered problems with RJE to an Amdahl 470. A document describing the problems will be included in the next tape distribution.

Intermetrics is acting as the sole US distributer for the Amsterdam Pascal Compiler.

They also have an improved UNIX accounting package, done entirely in the shell; it will be included in the upcoming software distribution tape #4.

Speaker 5 **UNIX at Fujitsu**

Yoshitaka Hiratsuka
Fujitsu Laboratories

As a result of problems with RK05 disks attached to their 11/45 running UNIX/PWB Fujitsu has developed a reliable disc system, the 40MB PF6032. It performs hardware level substitution of alternates for bad cylinders.

They have a LISP implementation running on their UNIX. It has no floating point support, and integers are +-32768. It supports 7000 free cells, and up to 200 literal atoms.

Their future plans for using UNIX are to port UNIX to new machines, including the M series (large) and the PFI500 (mini).

Speaker 6 **News from BBN**

Ben Woznick
Bolt, Beranek and Newman, Inc.

BBN runs a research environment, with a number of machines running a number of operating systems. In this kind of a mixed environment, the text editor can be a major problem, so they have developed a new screen editor. They also have a message facility amongst their systems. The "C machine" described last June is within design objectives; it now runs C programs, but not yet a full UNIX. They are currently bringing up ARPANET NCP code.

BBN is looking at issues in the remote maintenance of software, and developing a distributed UNIX system which will operate within a broadcast heterogeneous network. They have a UNIX-based network control center and network services manager for APRA-like networks, and are now offering

installation and support services on UNIX systems.

They are getting started on a C compiler for the Motorola M68000, and on converting the real-time O.S. "MAUS" to C for LSI-11's.

Speaker 7 **News from DEC**

Bill Munson
Digital Equipment Corp.

Bill is Senior Engineering Manager for the Telephone and Utilities Group at DEC. He described himself informally as a pseudo-product manager for UNIX (DEC doesn't distribute UNIX yet). He describes his function, in part, as trying to understand UNIX, so that he can pass on information within DEC and to customers. He hopes that DEC can avoid products which do not work well at UNIX sites, such as the PDP-11/60, PDP11/44, RL01's RK06/7's. He wants to ensure that UNIX migrates smoothly from the 11 to the VAX, and that users will be able to get the hardware info they need to be able to fully utilize that machine. He assured the audience that as new products are introduced, they will be field tested at UNIX sites -- probably Bell Labs. Field support at DEC is being provided with info about UNIX so that hardware maintenance and UNIX software types can coexist.

DEC has several UNIX licences, so they are able to study UNIX, and interactions with future hardware products. The main interest is in UNIX for the VAX since DEC already has enough OSs for the PDP11. DEC this year is spending more on UNIX than RSX11-M. DEC is working on a C compiler for VMS, as well as putting in some PWB features. VMS may even get bits written in C. The proposition that UNIX would loose its popularity if supported by DEC was put forward; since one of its main attractions is lack of manufacturer support. He said at one point, "People [outside the UNIX community] don't understand how easy it is to do things under UNIX". and "Watching people that are so dammed productive is amazing".

DECUS U.S. is almost certain to grant permission for a UNIX SIG; Mark Bartelt is coordinating this effort. DECUS CANADA and DECUS UK already have UNIX SIGs.

Speaker 8 **Distributed Ring Network**

Bruce Walker
UCLA

UCLA is doing network research, funded by ARPA; they have an ARPANET NCP program for UNIX, and are now working on a 6 system (11/45, 11/34 and VAX) ring network (8Mb/sec). The ring hardware is similar to that used at Irvine, MIT, and Logicon. At least two of the systems share a dual-port disc, and the network has access to an ARPA imp connection. The overall goals of the project are:

-network wide filesystem
-recovery software
-network inter-process communication
-distributed process families
-distributed database support

To enable the network wide filesystem, they intend to implement a global name space. Rather than base this on the syntax '/systemname/pathname', or '@systemname!pathname', they will be keeping 'filesystemname/pathname'. Thus a file has the same full name when viewed from any UNIX system. Filesystems will be replicated on different machines, by storing local copies of the inode list, and any

files in use. Updates will then be propagated around the net. Files will be brought into a system on a demand paging basis; some of the machines on the net may have no disc of their own, so pages will be brought in at ring speed, much like a disc driver would, using the system's buffer cache. Other issues to be considered are atomicity of file actions, synchronization, exclusive access, and recovery procedures.

---- *LUNCH* ----

WEDNESDAY JANUARY 30 AFTERNOON SESSION
Chairperson -- Don Ladermann

Speaker 9 **News From Western Electric**

Al Arms
Western Electric Co.

Al began by expressing surprise at the rate of growth of Users Group, and indicating that his purpose at these meetings is to answer questions, squelch rumours, and announce new products from Bell Labs and Western Electric. He indicated that the question of WECo continuing to licence software has been answered, for the time being. A recent decision was reached that software licensing is compatible with the Bell system's consent decree; so they will continue in the same fashion that they have in the past. The key appears to be that no support is offered and that the software represents merely a snapshot of a system in use at BTL.

Al expressed the hope that by 1985 the group will reach critical mass, collapse into a black hole, and finally suck in all existing card readers and punches.

There have been no new products since June 79, although some licensing changes have occurred. It is now possible to obtain a small business system CPU binary licence, which is based on the number of terminals supported, sliding from $750 to $9600. Bulk agreements can be arranged as well. A collection of popular PWB tools can be licenced in object form, including SCCS, for $3000 ($2500 subsequent).

Al then fielded a number of specific questions. UNIX tools can be port'ed to systems like RSTS only by licensing the target CPU for UNIX. Lint may be port'ed to PWB by adding a V7 licence to the PWB licence. PWB + V7 costs $42,000 for first licences. Mert will not be released. Contractors who work on UNIX systems must sign an non-disclosure agreement, or be licenced for the software products in use. UNIX-TS and UNIX-RT (Bell Labs' own internal versions) will probably not be released. LSX will not be released. WECo will try to release the Equipment Test Package that Bell Labs has for hardware testing. The kernel and utilities will not be licenced separately, since WECo is constrained in the way it operates, to sell 'as is'.

Speaker 10 **UNIX on the Harris /6 mini**

Bill Shannon
Case Western Reserve University

Bill described a project in which they port'ed UNIX V7 to the Harris /6 mini. He described details of the architecture of this machine, including the register sets, memory management, interrupt functions, and I/O. They encountered problems: the fundamental unit of addressability of memory is not a byte nor a word, types do not align to memory the same as they do on the 11, there is no stack facility in hardware, and the discs have an odd sector size. Nonetheless, they have V7 running on their 6 /6's, in swapping mode, and plan to convert that to a paging system.

Speaker 11 **UNIX on an IBM Series/1**

Paul Jalics
Cleveland State University

Paul described the Series/1 as a 16-bit mini, byte-addressable, with a reasonable instruction set, segmentation facility, and 8 stacks of segmentation registers. I/O is done through virtual addressing, there are some stack instructions, and the system has reasonable I/O gear. He indicated that IBM software was unusable. They did the port'ing without a PDP11. John Lion's UNIX commentary was most useful.

He went on to discuss some of the implementation decisions which were made along the way, and outlined the stages they went through in getting V7 up and running:

-obtained a cross compiler for C from U.of Delaware
-created a minimal UNIX filesystem on the system disc
-compiled and booted the UNIX kernel from the PDP 11,
via a communications link
-debugging cycles
-port basic utilities to the Series/1
-port the C Compiler to the new machine.

The implementation will be finished by next June.

---- *BREAK* ----

Speaker 12 **News from ISC**

Heinz Lycklama
Interactive Systems Corp.

ISC has a UNIX which runs under VAX/VMS, with most of the utilities. Heinz outlined some of systems and facilities ISC offers, including their own versions of mail, text formatting, screen editor, and a new device the 'UMC', which loosely resembles the KMC11. It can be used to turn char-at-a-time devices into DMA devices; the DZ11 terminal multiplexor is the classic example. Their VAX UNIX is a blend of V6 and PWB, and allows a smooth upward movement from the PDP 11. It also allows the use of DEC's DCL command language, and the shell interchangeably. It all seemed a little messy.

Speaker 13 **UNIX Performance Considerations**

David Mosher
Ampex Corp. (formerly of Berkeley)

David's concern has been with the development of tools for the evaluation of system performance, and determination of full system capacity. His presentation was a good overview of the areas which have been considered for improving UNIX system performance. Although seasoned hacks may have found this to be 'old hat', those newer to the UNIX game were delighted to receive such a coherent guide to performance consideration .

Swapping Performance
Swapping behavior falls into 3 categories: no swapping, easy swapping, and hard swapping. If the load is light enough for there to be no swapping, process throughput times rise linearly with the number of simultaneous tasks. In the easy swap area, only processes which are non-runable are swapped out, and the conflict for resources is still not too bad. Under hard swapping, the system may move runable processes out in order to bring other runable proc's in; but the proc's brought in may not actually run, for other reasons. By adding memory to the system, we can reduce or even eliminate swapping; with 1.5MBytes on an 11/70, performance is linear up to 40 users. David showed graphs in which he compared predicted performance with measured, before and after the addition of more memory. It was noted that the sudden rise in throughput time for a proc once hard swapping occurs has a large slope -- it is very nearly a wall! The swap algorithm can be softened somewhat by moving out large programs first.

The typical life of a process consists of:
fork and exec: which depend on the buffer cache, filesystem efficiency,
and read-ahead.
computation: depends on hardware cache memory, system overhead, and the
scheduler efficiency.
I/O: depends on read-ahead and filesystem efficiency
exit:

Filesystem Performance
It has been noted that the inode cache doesn't work very well. Two techniques can be used to improve it. Init can open /bin and /usr/bin to force those two inodes into the inode cache -- they will be used again and again. And schemes have been developed to use unwanted mode bits in the filesystem to indicate inodes which should be locked in the kernel.

The intent of block read-ahead is that when a block is read, the next block is read in preparation for its subsequent use. However, when the number of procs (or users or ports) is greater than half the number of buffers in the buffer cache, these read-aheads conflict and can result in a net deficit. Normally used blocks are put on the tail of the buffer cache; during a long exec, this flushes the entire cache. An improvement is to note that if the I/O aligns with the block boundary, put that block on the head of the cache -- it probably won't be referenced again, and can be reused. He claimed that the cache hit is 50-60%, or 1.5 blocks are read per seek to the disc. Note that keeping a filesystem 'linear' helps in this regard; this can be achieved by a complete dump and restor. A detailed discussion of optimal 'linear' layout can be found in the talk by Walt Lazear, Thursday morning session.

Suggested filesystem changes include keeping the root and tmp filesystems in core (costs memory and shifts the overheads), increasing the buffer cache, and extended buffer cache outside of kernel address space (eats up user memory and can introduce new overheads).

Processor Performance
Rescheduling is normally done when a proc blocks, or the clock says it time, and only in user mode. Thus the probability of a rescheduling = the prob. of being in user mode * the prob. of an event pending. Normally at 60hz, rescheduling occurs every second; by moving this up to 2 times or 10 times per second, a subjective improvement is noted, with 10 times per second somewhat better.

System Overhead
There is overhead involved in handling interrupts and system calls, with a minimum 320 microseconds to handle a system call. Terminal I/O is 90% output, only 10% input -- so pseudo DMA style drivers can be a big improvement, as can avoiding getc and putc. Wakeups in V6 are done through a linear search of the wakeup table, to its end. Noting the last used entry can trim this, and going to queues or a hashed lookup would be even better. SPL's exist in some places where they are not necessary, like in copyin and copyout, apparently to protect the supervisor registers. There was further info on this specific point in Dennis Ritchie's talk.

Speaker 14 **IDRIS -- UNIX V6 lookalike**

Mark Krieger
Whitesmiths, Ltd.

Whitesmiths has developed, from scratch, an operating system called IDRIS, which has the same system calls as UNIX (except Ptrace), the same filesystem, and is supposed to be exactly bitwise (binary) compatible with UNIX. They will have it available in an LSI-11, non-memory management version by Mar 21/80, with memory management by June 80. At the meeting last June he made similar claims that it would be available now. The markets they hope to compete in include LSI-11's and -23's which might otherwise run RT11 or RSX, turnkey products with UNIX-like systems inside, and packaged micro systems. The IDRIS system will lack the full range of utilities for awhile, such as YACC, TROFF, etc.

Speaker 15 **Contiguous Files in UNIX**

Mitchell Gart
Ampex Corp.

They have a technique for adding contiguous files to UNIX filesystems, in which a portion of a device is reserved for contiguous files, and the remainder is used for regular files. The motivation for this was a series of large image processing applications, in which a second port on the disc is used for transfers to and from hardware which isn't capable of deciphering the UNIX filesystem. Changes were required to mkfs, check programs, creat, stat, rm, open, close, write, etc. There were 7 changes to the kernel itself, each 2-15 lines. Plus the code to handle the files themselves.

Speaker 16 **Peformance Improvement: Large Buffers**

Jeff Schriebman
Jeff Schriebman Consulting

By increasing the size of the buffer cache to 100-500KB, and always reading 1/4 or 1/2 track from the disc, improvement in I/O throughput can be obtained in the 9-40% range at a cost of 15% of the CPU.

This modification took 4 pages of code, and is transparent to utilities. It works in V6, and should do so in V7. It affects only reads, as it is essentially an extended read-ahead. Jeff has a brief paper describing the mod.

Speaker 17 **Better Signal Management for IPC**

Paul Rubin
Advanced Business Communications, Inc.

UNIX application systems often need cooperating processes, one to provide operator control of a system, another to provide fast device control or run some computation. The traditional model is for one proc to write messages into a pipe or file, then signal the other proc, indicating that message(s) are waiting.

The problem arises that I/O is interruptable (non-atomic), and must be restarted following a signal catch. Vanilla UNIX allows signals only to be accepted or ignored; no delay or queueing of them is possible.

The solution Paul has developed is to add a 'pending' state to signals in UNIX. Thus 'signal(SIGHUP,-1)' will give the signal a pending state. The call 'sigs= await(fd,bitmask—of—signals)' will block until one of the signals indicated in the bitmask arrives, or return immediately a bitmask of pending signals, depending on the value of 'fd'.

This scheme allows for a more flexible and controlled handling of external events. It may be unecessary with the advent of V7 multiplexed files.

THURSDAY JANUARY 31 MORNING SESSION
Chairpersons -- Lou Katz and Sam Leffler

Speaker 18 **C and Pascal from Whitesmiths**

Mark Krieger
Whitesmiths, Ltd.

Whitesmiths has been involved in the production of C compilers and cross-compilers, for RSX, RSTS, RT11, the Z80, and 8080. They now have a VAX native mode C compiler and loader, and a Pascal to C translator. They are working on compilers for the M68000, 8086 and 370. Each C compiler product includes a C library. The Pascal will not run with any C compiler, due to the way it handles struct members.

Speaker 19 **UNIX on an Amdahl 470**

Gordon Kass
Amdahl Corp.

Gordon described the efforts by which they have port'ed UNIX V6 to an Amdahl system. They did this without the help of a PDP-11 to lean on and bootstrap from. They proceeded by obtaining Bell's C/360 compiler, the PWB C compiler, to which they added 470 code tables and their own assembler. Then by writing I/O supervisor drivers, and converting utility programs, they were able to support a UNIX under VM, alongside CMS, MVS, SVS, etc. This is a V6 system with the PWB shell, C compiler, SCCS, make, accounting, full screen editor, RJE, and an automatic disc fixer. Because it happens within VM, VM commands are still available; their system is allocated 6MB of main memory and 19000MB of disc. The standard I/O library has been modified for efficiency improvements.

In the coming months they will be converting to V7, improving filesystem backup, adding formatting to support Amdahl document prepartion, adding a debugger, C optimizer and networkir g. The software is not for sale; but if Amdahl customers are really keen on getting it, he suggested they ask their marketing rep's about it.

Speaker 20 **Optimized Disc Freelist Layouts**

Walt Lazear
Air Force Data Service Center

Vanilla UNIX mkfs and icheck -s build filesystem block freelists in a manner intended to optimized disc performance. Files will usually be allocated blocks sequentially from the freelist; and those blocks will be requested by the system in the same order. The first block is read, and following a fixed delay in the kernel, the second will be requested. If the freelist is structured to that the next block is allocated at the appropriate rotational distance, clean reads will result, with minimum rotational delay. The original scheme was based on an 11/40 running RP03's and is not suitable for other systems. To perform a measure for a specific disc and cpu, one needs specs for the disc, and a time measurement from a modified copy command. By way of example, an 11/70 system which had been attaining only 1 block/disc revolution was improved to 3/revolution, thus accomodating 50% more users. Other factors include individual processor speed, hardware cache, memory interleaving, bus configuration, superblock freelist size. Memory size, number of system buffers, number of mounted filesystems, user load and disc driver software are apparently not factors in this analysis (there is some disagreement on this point). Here are the magic numbers that Walt presented. Members of the audience suggested that they are conservative -- so try them out, but measure the results for your particular configuration. Also, these numbers do not apply directly to V7. The numbers are every—nth—block/sect—per—track.

DISC	34/35/40	44/45/60	70
RK05 J+F	3/24	3/24	2/24
RK05	5/24	4/24	3/24
RK06/7	9/66	7/66	5/66
RL01	8/40	7/40	4/40
RM02	13/32	10/32	7/32
RM03	20/32	15/32	10/32
RP03	4/50	4/50	2/50
RP04/5/6	14/22	11/22	7/22

RS03	10/16	8/16	5/16
RS04	19/32	15/32	10/32
RX01	1/6	1/6	1/6
RX02	1/13	1/13	1/13

Speaker 21 **Interactive Process Control**

Jim Kulp
I.I.A.S.A., Austria

Jim presented a number of problems which relate to the user's view of process initialization and control. The '&' feature is permanent, allowing no further tty input; the '!' escape is not general, requires an extra process, and may not inherit environment properly; ps must be run for background status; zombies are a hassle; the user is force to think of process id's, not job names; ignored signals can actually cause a swap in first; multiple signals are a problem; and interrupts can't be masked, only ignored or accepted.

To solve some of these problems, he has developed a set of control commands, whose syntax is as follows:
.f [job] run job in foreground
.b [job] run job in background
.s [job] stop job
.l [-l] list jobs
.q [job] suppress asyncronous reporting (quiet)
.k [job] terminate (kill) job
control — (underbar) suspend foreground process

```
for example,              du |pr -2 >> file
takes too long to wait, so <ctrl>—
                          STOPPED
                          % .l
                          i + STOPPED du |pr -2 >> file
                          % .b                    the job continues in background
                          etc.
```

In the facility, jobs are moved between foreground and background by changing their input/output from tty to /dev/null. Signals were changed to add sig—hold and sig—defer, which take signals and send them sig—hold, rather than the default. This code will be available in its V7 version.

Speaker 22 **USENIX Announcements**

Lou Katz
Columbia University

A 4th software distribution tape is being prepared; closing date for submissions is Feb 15, 1980. The tapes will be sent out about April, 1980, and will be sent to Institutional USENIX members. Arrangements have been made for interchange of tapes and newsletters with the UNIX user's Groups of the UK, Canada and Australia. There will be 10 newsletters coming out in 1980. USENIX elections

are coming up this spring; only Institutional members can cast ballots. The next USENIX meeting will be held at the University of Delaware, June 18-21, 1980. Proceedings will be published at that meeting, which should eliminate the need for detailed note-taking! Speakers should at least provide copies of their viewgraphs to the conference organizers. All material for publication should be camera-ready.

---- BREAK ----

Speaker 23 **Port'ing C to a Word Addessable Machine**

Sam Leffler
Case Western Reserve University

Sam describes his work as Yet Another Portable C Compiler; he has taken it to the Harris /6 mini, which he says is a very awkward machine with which to work (see Bill Shannon's talk, Wed afternoon). Pass 1 does all syntactic and semantic analysis; its intermediate language is expression trees. Machine dependent portions are translated, including all address offsets. The independent language is optimized. Pass 2 handles all other code generation. Expression trees are processed one by one, guided by a technique of estimating the number of registers used -- this is table driven. On th Harris /6 the registers are ugly (physically overlapping, some instructions imply register usage). There is no stack support, and pointers are inconsistent.

The project took 1-2 months of learning, 2-3 months of hard work.

Speaker 24 **Software Testing**

Bob Varney
Analytic Disciplines Inc.

ADI is working on programming of acoustic signal applications for the US Navy. There are several locations at which the work is being done, so they have set up procedures as an overall guide to software development. The basic concerns are software control, standard practices, and testing. All work is based on a 'Program Unit', which consists of documentation, source, test data input, test data output, and results analysis. These units are held online, and interactively updated. Static evaluation of programs is performed, and can be used in developing test cases.

Speaker 25 **ANSI BASIC for UNIX**

Chris Sturgess
Human Computing Resources Ltd.

Of the BASIC language processors available, ANSI BASIC nearly a proper subset of the BASIC's supplied by micro vendors, and UNIX BASIC intersects with these only slightly. HCR has developed a BASIC which includes the features of ANSI and micro BASICS, with extensions. These include string handling, TEK 4012 Graphics, block structuring, a help command, chaining (for running series of programs), appending for loading subroutine libraries, and exec for allowing the use of shell commands from within BASIC. The product is intended for use in teaching, business, and personal computing.

Speaker 26 **New CULC FORTRAN IV PLUS**

Robert Bradbury
Commercial Union Leasing Corp.
CULC's Fortran IV Plus is DEC compatible and MACRO-11 compatible. It comes with CULC's linker, and libr, and a converter from .o files to .obj files. It includes a UNIX system call library, and overlaying facilities. Recent improvements are an include statement, list directed I/O, the ability to mix C and Fortran, a compiler that runs 30% faster, and the code produced is 10% smaller, 15% faster than previous F4P compilers. There are 30 sites running this system, after 3 years on the market. There is minimal support on this product, for which binaries only are available; CULC customers must have a DEC Fortran IV Plus licence. This new version is now out, at $7500 commercial, $3500 educational, with a $750 charge .o upgrade to the new compiler.

Speaker 27 **Running V7 on small PDP-11's**

Bill Jolitz
US Geological Survey

A number of changes to V7 were required to get it to run on the 11/34. The bootstrap program runs in two stages; it needs separate I&D, doesn't support some discs, and needs at least 64KB of memory. A partial rewrite was required. UNIX V7 itself is too big. By reducing items in the parameter list (say to 12 buffers, 40-50 inodes), and removing accounting from the kernel, it can be made smaller. The machine assist, m40.h, has some bugs, including incorrect stack setup, a KDSA6 reference, and failure to enter at boot through trap properly (see m70.s). The RL01 driver doesn't handle raw I/O properly, and on the 70 it needs to allocate the UNIBUS MAP, which it doesn't. User programs which use floating point need to be recompiled, and some utilities have a size problem, as expected. ADB, DD, and AWK can be made to fit non-separate I&D, but LEX and F77 will be more difficult. Overlays are available in two flavours -- contiguous overlays in which there is a root segment and several overlay segments which are selected by a call to an overlay subroutine, and V7 has an overlaying function itself.

V7 bugs include the following. On a heavily loaded system, an inode misaddressing causes it to miss the start of files being read. The clist, if too small will crash the system. Increasing its size alleviates this problem, but comments from the audience suggested that a size change only postpones the problem, it does not remove it. The CU program has its arguments in a call to ioctrl reversed. In F77, longops routines use fixed point -- these can be changed to floating point.

Several persons promised to continue discussion of problems like these in subsequent issues of the USENIX newsletter.

---- LUNCH ----

THURSDAY, JANUARY 31, AFTERNOON SESSION
Chairperson -- Robert L. Cannon

Speaker 28 **SEED Database System**

Herb Edelstein
International Database Systems Inc.

Herb gave a description of the SEED Database System, which originally ran on a DEC 10, but now runs on ISC's UNIX. The product is claimed to be a portable CODASYL DBMS. It is based on ISC's Fortran compiler. The system addresses questions of security, storage management and concurrency. The price is $9500 for a binary licence.

Speaker 29 **Simplified DB Access: YACC and INGRES**

Nell Groundwater
Analytic Disciplines Inc.

ADI is working with the US Navy on software development for a collection of Advanced Signal Processors, built by IBM. There are 35 of these in use, each containing as many as 100 modules. The units are carried in vehicles, reconfigured as necessary, and shipped from site to site. Keeping the operation reliable and maintainable is largely a problem of keeping track of he array processors and their maintenance histories.

The solution that ADI has implemented is referred to as 'Decision Network Processing'. The user moves through a hierarchy of menus, and need not be aware that there is a UNIX system underneath. Different persons' responsibilities can be reflected in the menu choices available to their accounts. At a primary level of hardware/software configuration info, grep is used to find and present info to the user. At a secondary level, more complex requests are handled through the 'equel' query program of INGRES. The menu facility is available to other sites.

Speaker 30 **Geolab Interactive Environment**

James Herriot
US Geological Survey

James described their efforts to develop a collection of tools to allow users to interact with their data, performing transformations and producing plotter output. The data with which they are working are sometimes as large as 100MB time series samples.

Speaker 31 **Berkeley Virtual Memory UNIX V32**

Bill Joy
University of California, Berkeley

Bill described the mechanisms by which demand paging has been added to UNIX V32 (the original Bell version was a swapping system). The changes include two system calls, and 15000 lines of code (7500 of which are comments). Measurements were performed, using a synthetic user load, to measure the differences in performance under swapping/paging, 512byte system buffers/1KB buffers, separate disc controllers, size of buffer cache, and quantity of main memory.

At a load of 4 'user units' the swapping and paging systems performed similarly (computation bound), but by the time 16 units are running, the paged system runs twice as well.

Increasing the buffer size to 1KB improves performance of big jobs, at the cost of fairly complex kernel changes.

Increasing main memory improves performance overall, especially for giant LISP programs. Overall, changing from a small swap system to more memory, paging and big buffers gave an improvement of 13:31 down to 9:00 for a 12 job load.

Increasing the number of buffers in the buffer cache from 32 to 64 gave no net change to time required to run a test compile; PUT the total number of disc transfers was down 15%. It may not be noticeably faster, but it's easier on the disc.

The whole thing took about 1.5 man-years to do, shared between 2 people. This version of UNIX is being distributed; there are now about 12 sites running it. For info, contact

Keith Sklower
Computer Science Dept.
Evans Hall
University of California
Berkeley, California 94706

Speaker 32 **A Moderate Office DB Management System**

David Tilbrook
Bell Northern Software Research

David described his 'TIPS' system, which provides DBMS services on a scale between grep and INGRES. The system is based on a simple, standard method of text storage and output. It is currently has 30 users, and 70 databases, in the areas of data dictionaries, project management, bibliographies, correspondence, documentation, and telephone directories.

David suggested that TIPS' strengths were that it is based on simple text files, is small, easy to install, and requires no kernel modifications. He then admitted some weaknesses: it is slow at searching large bodies of info, it has no security beyond the UNIX filesystem (and no concurrency protection), is not quite fully bilingual (English/French -- it is partially), is based on and dependent on the QED editor, is not yet implemented using STDIO, and allows no nesting of structures.

A further strength, however, is that it is available to any UNIX site free -- it will be on the upcoming 4th software distribution tape.

Speaker 33 **Key'ed I/O Mechanism for UNIX**

Bob Lummis
Albert Einstein College

Bob has developed a key'ed I/O facility within UNIX, based on the facilities of the XEROX Sigma Series systems. The code will be sent for inclusion on the software distribution.

FRIDAY FEBRUARY 1, MORNING SESSION
Chairperson -- Wally Wedel

Speaker 34 **An Enhanced Spelling Checker**

David James
Ampex Corp.

Under the V6 spelling check, the text is broken into words, run through a unique sort and merged with the dictionary, to find exceptions. David has extended this to include a hash between the sort and the merge. Thus when looking for a word, its hash is used to form a disc block number, and the correct section of the dictionary if retrieved in at most 1 seek. If the block is already in the local cache, no seek is required.

Some general problems in spelling checking include: stripping hyphenation, capitals, and suffixes. There are no universal rules for suffix stripping, but it is possible to store valid suffixes with each word in the dictionary. For interactive use, the spelling checker suggests alternates when it finds a bad word, will perform source file updating as it goes, provide continuous dictionary u,dating, and handle automatic corrections for common misspellings. David has a 38,000 word non-proprietary dictionary which will be made available (this compares favourably with V7's 25,000 word dictionary). What he would like to collect is a list of typical misspellings of words, so that he can improve these techniques further.

Speaker 35 **Image Processing**

Bob Gray
Pattern Analysis and Recognition Corp.

Bob described their problems in developing a software development facility. Their applications are state of the art, and as such are not easily specified; there are many persons implementing new software; these persons want to share and trade code; and the end users are not the persons now developing the code. The system they have features iterative enhancement, provides online menus and documentation, and allows other processors to be down-line loaded. It does this through a collection of 75 shell command files, which make extensive use of release numbers and updating. This ensures that there is always a working version of any application, and leaves more time for image processing development.

Speaker 36 **3D Image Processing for Biology**

Noel Kroff
Columbia University, Biology Dept.

Noel is using 3D vector graphics to study and model biological structures. This work finds application in the following areas:

- molecular modelling, protein stucture and interaction

- design of antibodies through recombinant DNA
- crystallography
- reconstruction of neural systems

To facilitate the last area, they have a laser scanner, called the 'ANT' (automatic nerve tracer), which allows them to look at images in a reasonable fashion. Noel showed slides of various examples of their work.

Speaker 37 **Status of UNIX V7**

Dennis Ritchie
Bell Labs

Dennis summarized the bugs which have been reported for V7, and emphasized that the Newsletter will be used for reporting any further bugs, and the fixes.

1) in low.s, the parity fault interrupt is set wrong, at 7 instead of 10.

2) in converting V6 filesystems to V7 with TAR, there is a sign extension problem in the stat routine.

3) when running an 11/70 with RP discs on the UNIBUS, the bootstrap doesn't set up the busmap. As well, the RP03 doesn't allocate the busmap.

4) wakeup() calls setrun() which recursively calls wakeup(). wakeup() should go back to the beginning of its list, rather than continuing from where it is.

5) ldiv, lrem, aldiv, alrem in libc perform long divisions wrong, as when dividing by 0100000.

6) adb displays floating point registers incorrectly (in the wrong order).

7) the tm tape driver has an incorrect define for the RLE bit.

8) the C optimizer can go into loops; this is known to happen for long compares.

9) ioctrl was written so as not to flush the input queue when switching tty modes. This can cause problems between normal and cbreak modes. The simplest fix would be to force the input queue to flush.

10) the problem with a small clist crashing the system may be related to the presence of the DZ11 driver, which seems to be a common feature in those sites which have reported this bug.

11) TU16 and HT's can hang, with the interrupt never occuring. A timeout could be inserted to check for this type of condition.

12) the removal of spl5() in copying and copyout should be done with care, it may be needed on the smaller 11's which don't have the supervisor registers.

13) in relation to the PDP-11/44 not running UNIX properly, V7 does not use the extra register set, so there should be no problem.

14) text.c has a parenthesis problem; running lint -h on it will show it up.

Speaker 38 **West Coast Users Group**

John Bass
Onyx Systems

The West Coast Users Group is accepting applications from individuals for membership; it is open to those interested in the ideals of software development environments such as UNIX and Software Tools. Its general goals are to interchange technical information, with primary focus on activities which are in the public domain, and to foster interchanges between groups of users of proprietary packages. Newsletters will be published, containing short papers and letters to the editor. Regular features may be run as well, such as hardware and software reviews. This users group is looking for volunteers to help in the work.

Software distributions will include the earlier conference tapes, should be done on a 2-3 week turnaround from receipt of order, will be sent out only on new tapes, at a cost of $20-35 per tape (tape+handling+mailing).

To become a member, send $10. US to:

Login West
PO Box 581
Menlo Park
California 94025

Speaker 39 **Interactive Image Processing**

Rex Tracy
TASC

TASC is working in areas of image processing, including mapping, geodesy, charting, navigation, image data extraction, geothermal sources, mensuration, remote sensing and oceanographic imaging. They are in the business of evaluating architectures and algorithms for use in these areas. They have an image processing language under UNIX PWB, which they will be moving to a VAX. Their system uses an FPS AP-120B array processor.

Rex showed slides of images in various stages of enhancement, including the creation of elevation contours, gravity anomaly fields, edge enhancement, and localized histogram equalization within a user-sketched area. He also showed images on which an automatic classifier had been applied, to separate regions of river, city, forest, and other vegetation. A primary motivation in this work is to prepare images in such a way as to aid human photo interpreters to do their job.

He went on to present a digital imagery database over a small region, built up of a photo, map, railroads, transmission lines, and cultural information.

The graphics tools they are using are apparently public domain (they obtained them from another site), but the image processing code will probably not be distributed.

Speaker 40 **Interactive Graphics in Pharmacology**

Tom Ferrin
University of California at San Francisco

Tom's work is intended to support research into rational drug design, which they hope can replace simple discovery of new drugs. This effort requires some understanding of the interaction of complex molecules with human tissues; the computer graphics lab provides a means by which researchers can interact with molecular models containing 10,000 individual atoms. To perform the same task with physical models would require tons of modelling materials. Details of their work can be found in the winter issue of the journal 'Computer Graphics', a publication of the ACM SIGGRAPH.

The specific tools which support these efforts include:
A graphics subroutine package for the E&S Picture System 2, a beam-penetration, stroke-writing system costing $50,000+.
Modifications to support one real-time process. Before the mod's, average time to wake this one proc up was 30-40 ms, with a worst case of 120ms. Now it is 5ms average, 20ms worst case. To do this, the proc is locked in core (preferably at the high end), and long kernel modules like mvcore are changed so as to be pre-emptible when the real-time proc needs to be scheduled.
Support for a variable number of subroutine arguments within separate I&D space programs. This involves a mod to the 11/70 processor, and has been documented in one of the older newsletters.
Fast I/O, using raw contiguous files.

Tom screened a film which they made straight off the face of their display station, showing their interactive tools in use. The particular molecules being handled in the demo were a large transport molecule being used to carry a smaller growth hormone, as might be used in the treatment of premature babies. Tom indicated that the 3D vector data is passed through the matrix hardware twice, once to display it, and once to calculate the intermolecular bond distances, which are display continuously as dashed lines, and numerically. The hardware supports hither and yon clipping planes, which allow the user to limit the view of the molecules to an area of interest partway down their total depth.

Tom agreed to send out the mod's he has made to UNIX.

Speaker 41 **Speaking Terminal for Blind Programmers**

Ronald A. Morford
Drug Enforcement Administration

About a year and a half ago, industry had yet to provide adequate tools for blind persons to use computers; braille terminals suffer from low reliability, due to their mechanical nature. Ron set out to develop his own voice output terminal, called VERT (VERbal Terminal). VERT is an 8080-based unit which is placed on the tty line between host and standard terminal. It holds 16KB of PROM for the rules of pronunciation (after McIlroy's techniques) and user interface, plus 4KB of RAM for buffers. Information is buffered, and spoken using a Radio Shack VOTRAX voice synthesizer. Ron has the unit running at about 90-100 words per minute; his goal is 300 WPM, but not with this hardware configuration.

The user can operate VERT in a variety of modes. One key at a time input mode can be used for echo, and to determine variations in keyboard layout (hunting). Word at a time has normal uses, while no echo on input is used for entering large amounts of text. Output modes vary as well, depending on the context. The unit can be set to buffer a character at a time, word, line, or paragraph. It has been

taught a mode in which it understands programming statements, and will present programming elements more explicitly to avoid ambiguity.

Ron played a tape of some typical interactions, and indicated that he concentrates on writing code which 'sounds' good. Other programmers' code is often hard to understand when spoken. He feels strongly that any blind person who can use such a system should be given access to one, and university, government and industry should should be doing it. There is a company which is developing and selling VERT 3000 terminals, for about $5,000:

Automated Functions
672 Armistead St.
Alexandria, Virginia 22312
(202) 362-0979

Ron's talk was followed by a number of questions and comments. There are other groups developing tools for the handicapped, and one unsatisfied need is for a portable note-taking system.

FRIDAY FEBRUARY 1, AFTERNOON SESSION
Chairperson -- Ed Szurkowski

Speaker 42 **Summary of Tuesday Software Tools Meeting**

Debbie Scherrer
Lawrence Berkeley Lab.

Debbie presented a brief summary of the Software Tools Meeting. She also presented an amusing quote attributed to R. Reich from NYU:
"The day to day travails of the IBM programmer are always so amusing to those lucky enough to have never been one -- like watching Charle Chaplin cook a shoe."

Speaker 43 **TTY Handling**

Dan Franklin
BBN

BBN has an extended terminal driver. It allows setable control characters, a polite mode in which incoming messages are interleaved with keyboard activity, deferred echo, 8 bit mode, page length parameterization, and so on. The code can be distributed if there is sufficient interest. The question was raised as to what terminal control character conventions people use. Roughly 30% of the audience uses the original Bell conventions, 30% have following the DEC standard, and 40% are using something else.

Speaker 44 **Putting up UNIX when you don't know much**

Robert A. Morris
University of Massachusetts, Boston

Robert presented a brief history of the evolution of his new UNIX system. His conclusions were that new sites should seek some help from within the UNIX community, and that having a friendly site which will make a bootable copy of your system media is a big help.

Speaker 45 **STDPLT: A Device-Independent Plotting Package**

John Nickolls
Ampex Corp.

John described an output-oriented device-independent plotting subroutine package which they have developed. The package will be on the upcoming 4th distribution tape.

Speaker 46 **Two LSI Layout Tools**

Robert Mathews
Ampex Corp.

Bob described tools they have developed for LSI layout design, which relate well to courses based on the book 'Introduction to VLSI Systems', by Mead and Conway (Addison-Wesley 1980). The user passes layout commands from RATCIF through CIF to a plotting routine called PEN, which draws the layouts. Different dot patterns are used to display rectangular regions at various layers in an LSI circuit; these stipple patterns are chosen so that even when overlayed, it is possible to identify underlying layers. The system produces raster output, and can be used with Gould and Versatec plotters.

Speaker 47 **SCANARGS: A Command-line Argument Scanner**

Gary Newman
Ampex Corp.

In the collection of tools in the UNIX system, there exist a number of different conventions for the expression and parsing of command arguments. The motivation for the SCANARGS subroutine which Gary has developed is to improve the human interface to these tools by encouraging uniformity of argument syntax, while allowing clean recovery from errors in argument lists, and allowing this code to be shared. Thus, 'usage: ...' messages will be handled automatically from the format description passed to scanargs().

The subroutine call looks like:

```
scanargs(argc,argv,format[,pointer]...);
```

where elements in the format are similar to those in printf and scanf. The usual '%' character indicates that an argument is optional, while using '!' indicates a required argument. For example, the syntax for diff is

```
diff [-b] -[efh] file1 file2
```

for which the call to scanargs would be:


```
scanargs(argc,argv,"diff b%- efh%- file1!s file2!s",
                &blanks,&flags,&file1,&file2);
```

Briefly, this means b is an optional flag to be stored in blanks, a second set of up to 3 optional flags should be stored in flags, and the file names (required) should be stored in two string areas.

This facility is based on STDIO, and will be on the 4th distribution tape. Dennis Ritchie commented that there is a similar package in use at Bell Labs, and agreed that this is an important area of discussion, which will be continued in the Newsletter.

Speaker 48 **Black Holes In UNIX Filesystems**

Joe Yao
Science Applications Inc.

Joe described a situation he encountered on his system, in which a directory had grown in size to the point where its size had overflowed. Thus it has reached 'critical mass', and files placed in it simply disappeared. His attempts to find or remove the collection of files in that directory failed, although the filesystem space was indeed allocated. Rather than take the system down long enough to patch the problem, he decided to try to to patch it online -- which he did successfully. His talk led to an animated discussion of a variety of similar situations which can arise, and how they should be controlled.

Speaker 49 **Invitation to Join an Informal International UNIX Network**

James Ellis
Duke University

People at Duke, and at the University of North Carolina, Chapell Hill, are forming an informal UNIX net, in which they are inviting participation. It will, at least initially, be based upon the uucp code from V7, plus modified mail and news programs. The architecture of the network was not spelled out, although it would appear that it will initially be a star, with Duke at the center. For information, contact:

James Ellis
Dept. of Computer Science
Duke University
Durham, NC 27706
(919) 684-3048

Speaker 50 **Disc Scheduling AND V7 on a Z8000**

John Bass
Onyx Systems

There are at least four general techniques which can be applied to disc scheduling in disc drivers: FIFO, sort the requests in the IN direction, sort them in the OUT direction, and sort them in IN and OUT, depending which way the disc heads are moving. John suggested that read-ahead will work well only with IN ordering, and that IN ordering is preferable for that reason. The norm in UNIX drivers is IN and OUT ordering.

Onyx has developed a table top Z8000 system which runs V7 UNIX. It is available with a binary licence, plus language processors for C, Pascal, Fortran, Basic, and Cobol. He suggests that its speed may be comparable to an 11/45 with DZ11 and RP04, plus or minus 20%, depending on the application. It now supports 512KB main memory on an 8 user system, with 10MB disc, at $21,700 for the machine, $2500 for the V7 binary. Single user systems should be available for about $20,000, and peripheral improvements are on the way.

Speaker 51 **C on the M68000**

Charles Forsberg
Sidereal Inc.

Sidereal is involved primarily in the development of microprocessor based terminals for use with TWX, etc. These are M6800 and M68000 based, and they have port'ed the Ritchie C compiler to the M68000. Their assembler is based on the University of Queensland 6800 assembler. There is, of course, some question as to the relative merits of the M68000 and the Z8000; Charles prefers the M68000, in part because is supports a 32 bit address space. A test program in C which took 96 bytes, and 4.3 seconds on an 11/45, took 114 bytes, and 5.1 seconds on the M68000. They will be marketing this C compiler, although the price is not yet known.

All sessions ran close enough to the correct time that no Saturday session was required.

UNIX Performance
An Intraspection

David A. Mosher

Ampex Corporation
401 Broadway MS 3-59
Redwood City, CA 94063

ABSTRACT

This paper presents models and reasons for native UNIX performance. Some figures are given for real systems. Some improvements are mentioned. The purpose of this paper is to give installations a tool for evaluating the performance of their systems and sufficient information to know when their system has reached full capacity.

The results presented in this paper are based on years of analysis of the UNIX systems at the University of California at Berkeley. These results are generally applicable though the order of importance may change.

Introduction

To begin with, I feel that is important to observe that there seem to be two different definitions of ''time sharing''. One definition implies that time sharing is the sharing of resources by users which would otherwise not be fully utilized by a single user. The other definition implies only that resources are shared regardless of whether the resource is already fully utilized.

The consequences of adopting either of these definitions will become clear in the pages ahead.

Performance

Performance can be measured in terms of I/O, swap, comutation, and overhead. I/O is defined as the amount of blocks tranferred to and from the file system i.e. exec blocks and blocks read or written by the user. Swap is defined as the amount of movement of processes in and out of main memory. Computation is defined as the amount of central processing time being used by the user community. Overhead is defined as the amount of central processing time being used by the operating system.

In the following sections, the four areas of performance will be looked at in-depth.

Swap

Excessive swapping degrades a system more quickly than any other area of performance presented in this paper. This is the reason for discussing swapping first.

There are three basic levels of swapping. The first and easiest case to look at is no swapping. If processes are not being swapped in and out of main memory then processes execute at the speed of the memory and are governed by the scheduling of the central processing unit. Thus, a performance curve (execution time of a single process versus the number of simultaneously executable processes) would be a linear increase in time as the number of simultaneously executable processes increases.

The next level of swapping is sometimes referred to as easy swaps. (An easy swap is when a sleeping process can be swapped out and an executable process can be brought in.) At this level of swapping, the cost of a swap (e.g. the time to reload a process into main memory) grows from small to a large factor in comparison to the process's life time. As the number of swaps per time period increases, the cost to each process grows as the size of the process times some factor over the execution time allowed between swaps.

A few typical numbers for the factor mentioned above are given below:

RS04 4.0 + 0.66 = 4.66
RM03 1.6 + 1.16 = 2.76
RP06 2.5 + 1.16 = 3.66

All numbers are in microseconds

The first number is the typical tranfer time of a word from these disk systems to main memory. The second number is typical arm movement latency and rotational delay calculated on a per word transferred basis.

In general, the number of swaps in a time period will be a function of the number of simultaneously executable processes.

Thus, the performance curve changes slowly from the slow rising line of simply sharing the central processor towards a rapidly rising line governed by the factors shown above.

At some point as the number of simultaneously executable processes increases, there will be no sleeping process in main memory to throw out. So executable process will have to be thrown out of main memory to allow other executable processes a chance to be swapped into main memory. This is sometimes referred to as hard swapping.

Since processes are guaranteed to be in main memory for a certain amount of time, once in, there will be a limit to the number of swaps per time interval. Thus, performance at this level again depends linearly on the number of simultaneously executable processes but at a much accelerated rate.

Mathematically, these levels can be viewed as the following equation:

Time = (factor * size) * swaps(N) + (execution time)*N

where N is the number of simultaneously executable processes

swaps(N) = ((n>MAX ? (1-MAX/n) : 0)
+ (N>MAX ? (1-N/MAX) : 0)) * U

where MAX is the maximum number of processes in main memory,
where U is the characteristic usage of the system
and where n is the number of processes

The following figure is a model based on a simpler version

of this equation. The two lines in this figure represent the assumtotoes to the expected performance curve.

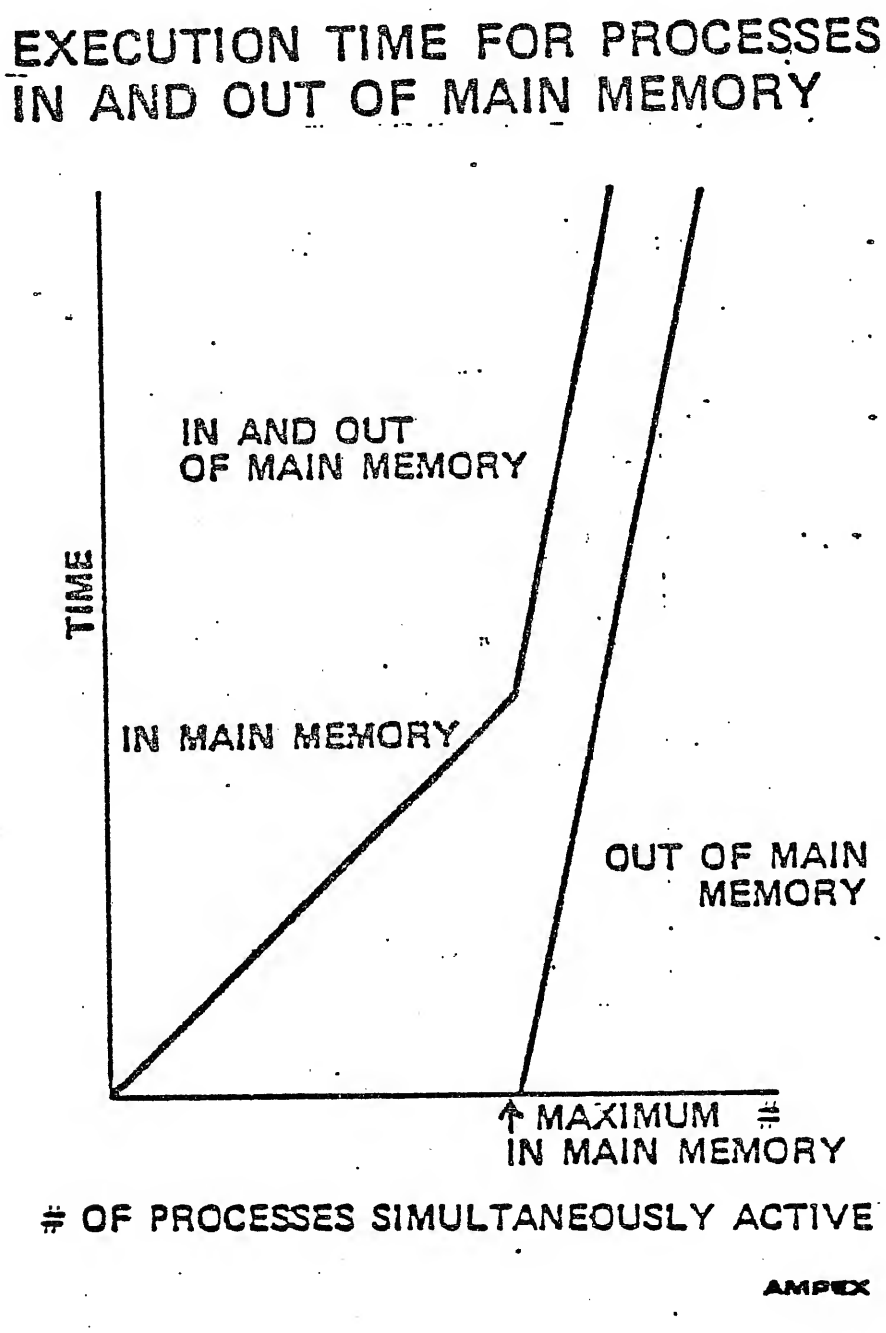


Figure 1

A few examples at this point will help to clarify the issue of swap performance and show validity to the models and reasons given above. Below is a performance curve for a UNIX system running on a PDP 11/70 with a RS04 fixed head disk for swapping and another RS04 for the root file system. Both of these drives are on the Massbus. The user file systems are on several Cal. Comp Model 615 drives. These drives

are on the Unibus. This system has 72 ports of which 38 are publicly available and 8 are dial up ports. The rest of the ports are used privately.

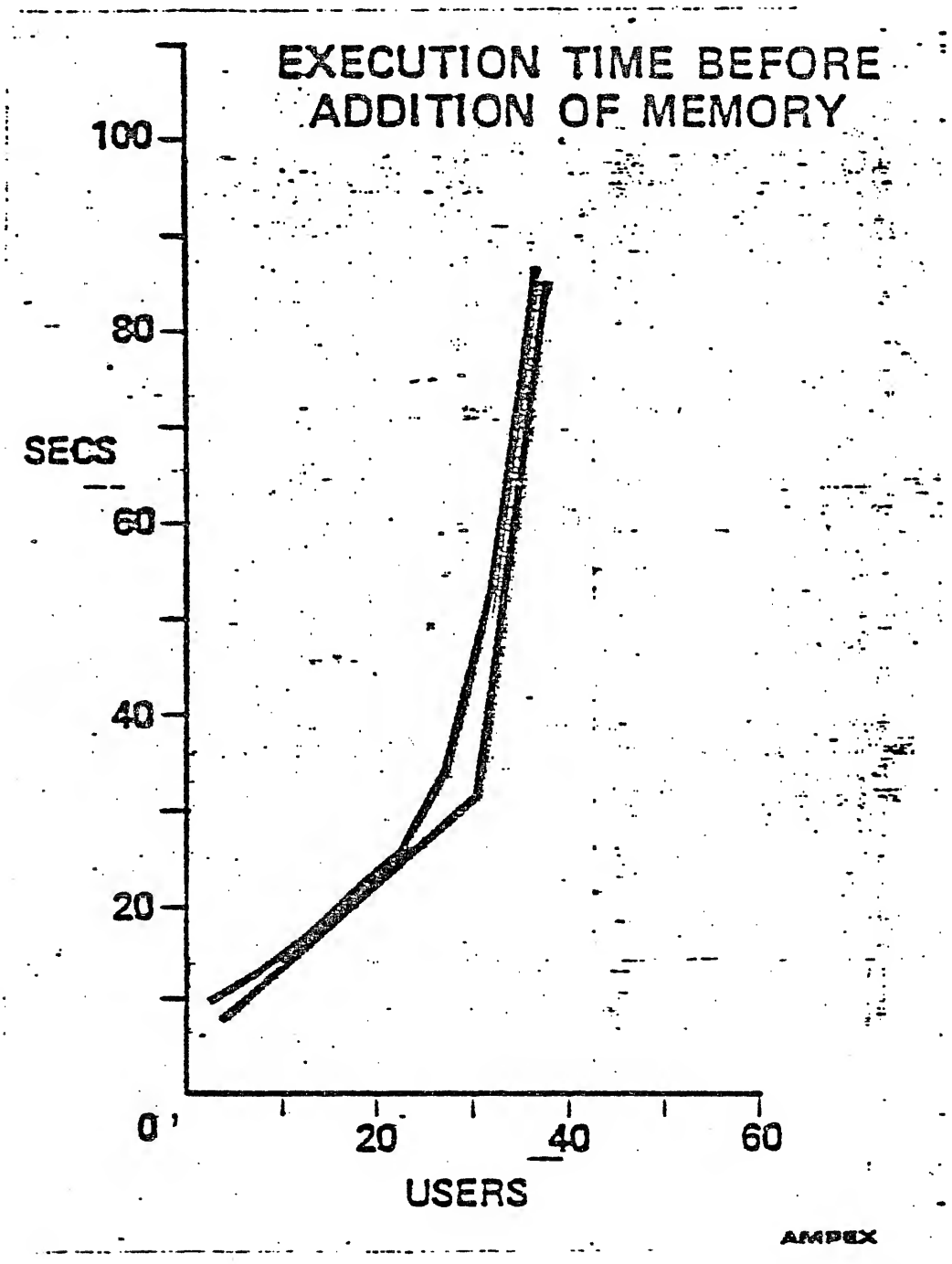


Figure 2

As can be seen above, the model described for swap performance fits this performance curve fairly well. After these measurements were taken, the system nad a major addition to its capability. The system had previously been equipped with only 512k bytes of main memory. An additional megabyte of main memory was added to this system. The addi-

tion tripled the amount of memory available for user processes. Below is the performance curve for this system after the addition.

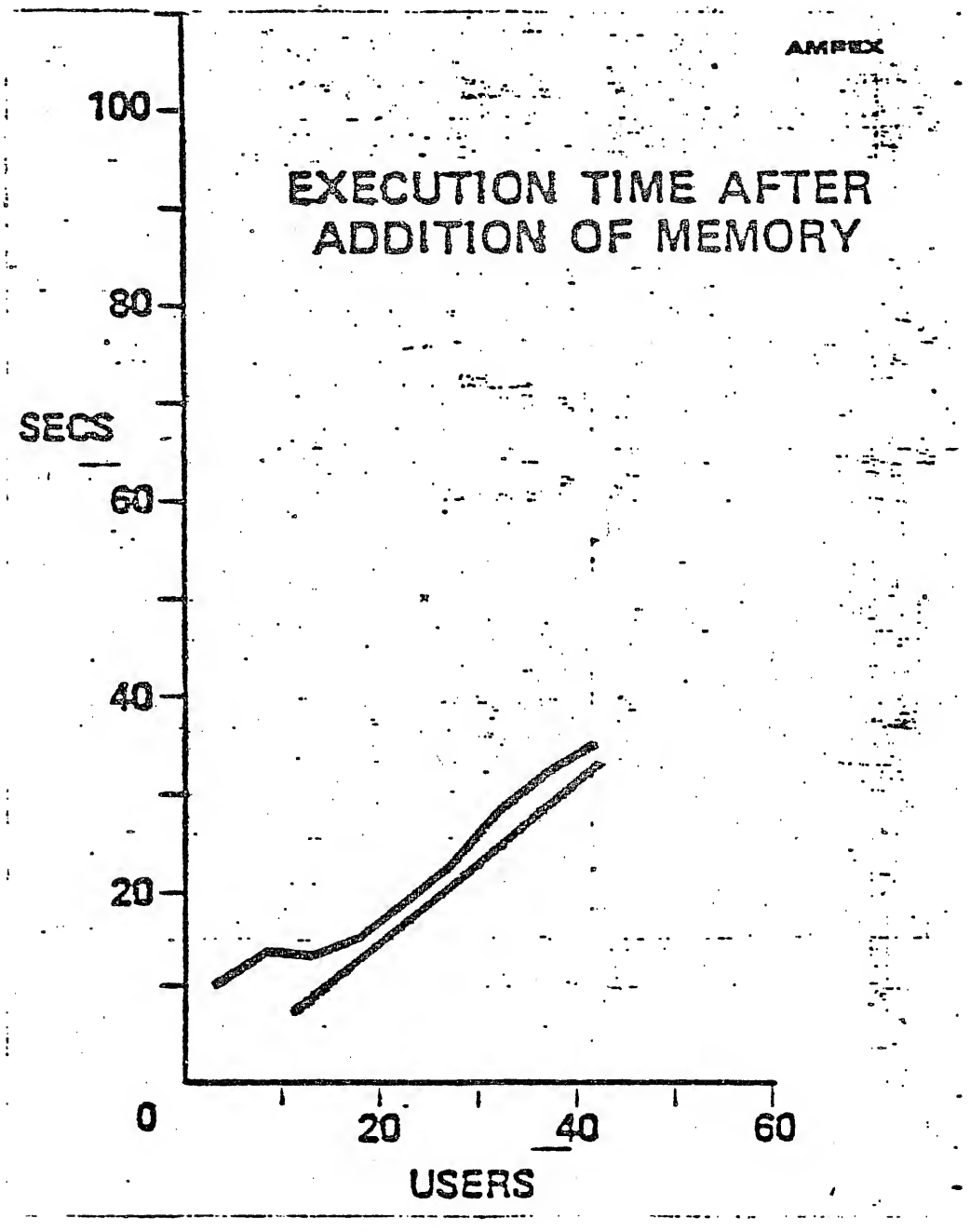


Figure 3

There are two important differences between these performance curves that should be noted. The swap degradation has vanished. Measurements of this system showed that before the additon of memory, the swap scheduler was running 100% of the time at maximum user load. (Note that load was a function of what users were willing to put up with as far

as response time was concerned.) After the addition of memory, measurements showed that the swap scheduler was running less than 10% of the time. This activity can be explained by the movement of shared text from the swap disk to main memory.

The other difference to note is that the performance curve has moved down along the time axis in figure 3 even though the slope remained the same. This change is due to the fact that the new memory was almost twice as fast as the old memory.

Even though the PDP 11/70 has a cache, the increase in the speed of the main memory had a number of effects. Cache misses take less time. Direct memory accesses take less time to complete. The central processor could get more memory cycles between direct memory accesses.

The left side of the performance curves of figures 2 and 3 start out together. The reason for this similarity is due to file system performance. (File system performance will be discussed shortly.)

From these two examples, we can see that the amount of main memory and the speed of the main memory are very important to system performance.

Even though these figures are based on the characteristics of the users of this system, the model still applies to any system. Also remember that the performance curve will be pushed up the time axis by a factor of two approximately for systems without a cache.

In general, the amount of main memory for best performance is the number of simultaneous users (the number of ports is adequate) times the average size of a process plus the size of the operating system (a number to start with is 32k bytes per process).

In closing, I would like to share two other pieces of information.

If you are not fortunate enough to have enough memory for your user load, you may wish to look closely at the algorithm for the swap scheduler. This algorithm has the ugly feature of allowing small processes to stay in main memory almost indefinitely. These small processes will have the tendency to fragment main memory and cause a loss of available memory for user processes. This ugly feature can be partially corrected by adding in the time a process has been in main memory times a factor (8 is fine) to the size of a process during the determination of the largest processes that can be easily swapped out.

Measurements of memory fragmentation show that 10 percent of the memory is lost to fragmentation anyway.

The question of a fixed head disk versus a moving head disk for swapping is commonly asked. Let me answer this question with the following measured facts. I have yet to see a moving head disk drive transfer continuously at a rate above 100 blocks per second when used as a swap device.

I have seen a fixed head disk (of a similar transfer speed) transfer continuously at 500 blocks per second when used as a swap device. The only conclusion I can draw from these measurements is that any seeking destroys system performance.

In conclusion, I would say from my measurements that swapping should be avoided for good performance.

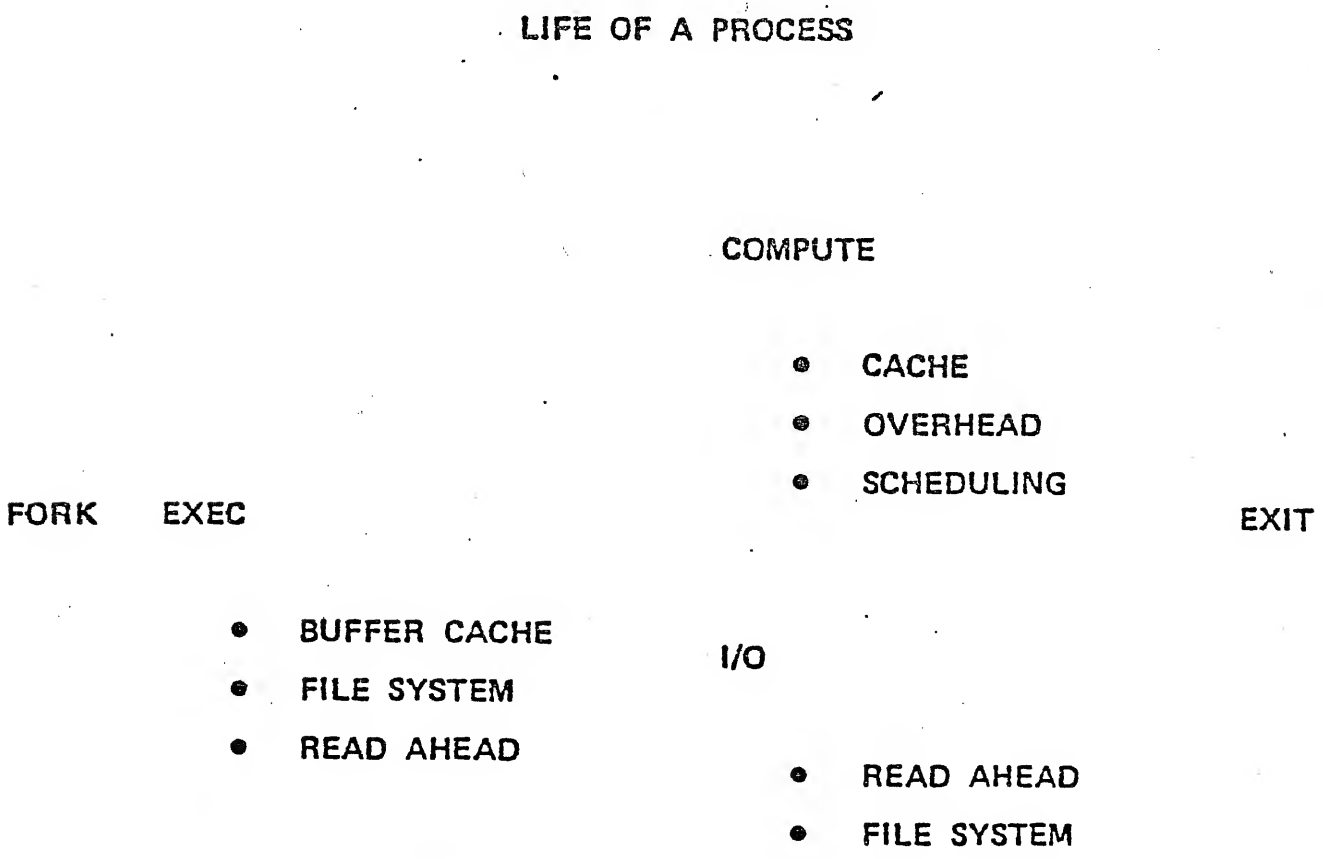


Figure 4

User Defined Performance

A proper perspective on computation and file system activity is needed before considering performance effects in these areas. The following discussion and figure should provide an adequate perspective.

Let us for a moment consider the life of a process. Birth usually begins by a fork system. Since forks depend upon memory space or swapping, the speed of a fork is determined by swapping activity. A fork is usually followed by an 'exec' system call. An exec loads the image for a process either totally from the file system or partially from the swap disk and partially from the file system. Beyond this point, the life of a process is determined by the user. The user may or may not read and/or write data in the file system. The user may or may not do a considerable amount of processing. Hopefully at some point, the process will exit via one method or anotner.

As can be seen from figure 4 above, we must look at exec's as well as user file system activity to see the whole picture on file system activity.

File System

The first file system activity from a process will be a path search. The basic file system activity during this search is seeking between the storage location of the inodes and the storage location of the blocks that make up the entries of a directory. This search takes more time as the file system grows in depth and bulk.

Path searching may be done rarely but it is expensive in its use of the disk system. Searching is sped up by having the directory inodes already in the inode cache. The directory may already be in the inode cache if more than one process is using it.

The probability that a directory inode is in the inode cache is small due to the rarity of searches except if the directory is the current directory.

To improve on patn searching performance, a number of approaches have been tried with varying success. One approach is to open directories used frequently by a program like 'init'. Another approach is to have the 'set user id' bit mean that the inode is to be locked in the inode cache on first use until the device is unmounted.

Measurements of directory inode activity show that 50 percent or more of all activity is through /, /bin, /lib, /dev, and /tmp and only 50 percent or less is through the rest of the file systems.

These solutions are not elegant but do provide some improvement in file system performance. The inode caching algorithm itself may need to be improved if a more elegant solution is desired.

Once the exec has the file from which the image is to be loaded, there are two possibilities. Either all or part of the image is read from the file system. If part of the image is loaded from the swap disk, the speed at which it is loaded is determined by swap activity.

How quickly the image loads from the file system depends on its linearity and the effectiveness of the read-ahead. The linearity of the image is dependent upon the linearity of the free list of that file system and the speed at which the free blocks are used.

Only in version 7 has anything been done in this area to ensure linearity on a dynamic basis. Occasionally for version 6 systems, static but very active file systems should be dumped and restored on a freshly made file system to improve the linearity of files commonly used.

The effectiveness of the read-ahead in UNIX systems depends on the buffer cache performance. Specifically, the number of buffers versus the number of users governs the read-ahead effectiveness. In order for read-ahead to be effective, there must be two buffers available: one for the currently needed block and one for the read-ahead block. In fact, if insufficient buffers are available, an attempt to do read-ahead may delay the currently needed block from being read in. This relationship between the number of buffers and the number of users doing file system activity is illustrated below.

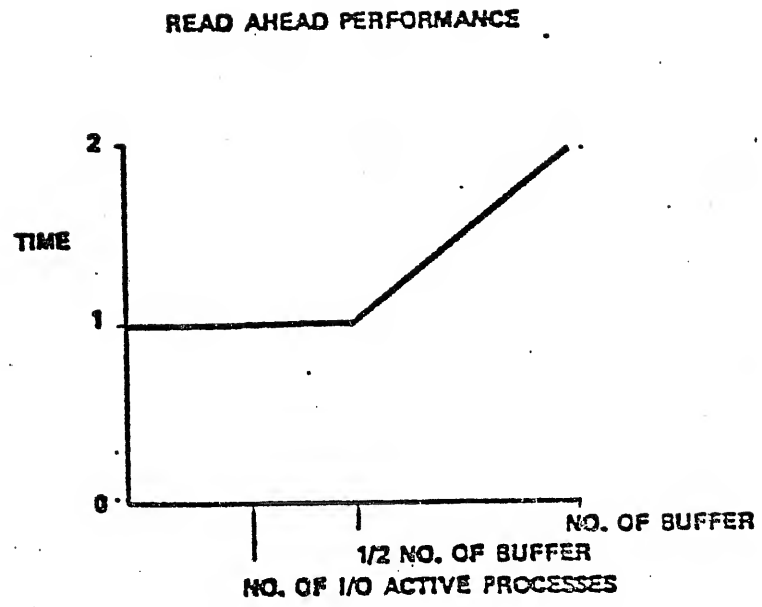


Figure 5

As a rule of thumb, the number of buffers a system needs is twice the number of simultaneous users to guarantee effective use of the read-ahead and good file system performance.

Note that all of the above also applies to user file system activity.

Buffer Cache Problem

The loading of an image during an exec system call unfortunately has a negative effect on user file system performance.

In original version 6 systems, the image being loaded would cause every buffer to be used because the buffers were put on the end of the buffer free list. Thus, any partial file system activity by other processes will be flushed from the buffer cache. Read ahead blocks for other processes are also flushed out as well.

This problem may have been fixed in version 7 though the solution chosen needs further investigation. The main idea of a solution is to put buffers unlikely to be used again at the beginning of the buffer free list. Definition of unlikely is usually based on the same assumptions that read ahead is. Thus, a buffer whose contents has been read or written to a block boundary is assumed not to be used in the near future.

To get a feeling for file system performance, let me throw out a few figures.

Measurements show that only 1.5 to 1.6 blocks are transferred per seek. This measurement coincides with the hit rate on the buffer cache of 50 to 66 percent. These measurements might mean that the only other block read between seeks is its read-ahead block, if that.

Obviously since blocks per seek is small, the time per seek becomes important. A typical time appears to be 20 milliseconds. These measurements are supported by the fact that file system activity only amounts to 10 to 20 blocks per second maximum. This rate is in contrast to 100 blocks per second for moving head swap devices -- a factor of 10.

There are a few other ideas for improvements in this area I would like to comment on below.

The question has been asked if the root or tmp might be better off placed in main memory as a special file. This idea is only reasonable if there is more memory available above and beyond what is needed for processes to run without excessive swapping and if there is lots of central processing time to waste. There is so little processing time available on most systems that this improvement is only feasible for special purpose systems.

Another improvement to accommodate more buffers is to overlay buffer space or move the buffers totally out of kernel main memory. Here again, it is important that there be no loss of main memory available for user processes. Overlaying buffer space is not expensive as far as central processing time but this scheme is yet unimplemented. Moving the buffer space totally out of main memory will cost considerable amount of central processing time or delay in processing interrupts.

In conclusion, I would like to suggest that systems should fit on suitable hardware rather than playing games to make it fit on a smaller machine. Good performance will only come from a system with proper hardware.

Computation

A user may use as little or as much central processing time as may be needed. Obviously, central processing time can be used 100 percent of the time by a process which just loops forever. The main objective in this area of performance is to share the central processing unit fairly. The question is what is fair and what will fairness cost in the area of operating system overhead.

Even to consider scheduling the central processing unit

fairly, some mechanism is needed to interrupt the process which has been using the central processor. Rescheduling is currently requested during clock interrupts -- once a second -- and during interrupts for completion of transfers by devices. Rescheduling is not done unless this interruption occurs while the user process is running. Thus, the percentage of time the system is in user mode is important to how often the central processor is rescheduled.

Mathematically, the probability of rescheduling is the product of the probability that the system is in user mode and the probability that a request for rescheduling is pending.

P(reschedule) = P(user) * P(request)

For now, let us harshly consider rescheduling probability as a measure of fairness. Then fairness should increase if the amount of 'user time' available is increased or if the number of requests is increased. In the first case, user time is gained by reduction of operating system overhead. Thus, any solution which increases overhead is of marginal, if any, benefit. In the second case, there are two ways to increase the number of requests for rescheduling. One way to increase requests is to increase the number of completion of devices somehow. This approach increases operating system overhead so it is rejected as a solution. The second way is to increase the number of rescheduling requests generated by the clock interrupt. Since the clock interrupts sixty times a second, rescheduling requests can be done more often than once a second. The overhead here is nominal since the state of the process has already been saved and context switching is relatively inexpensive.

In general, measurements show that most systems spend about 50 percent of its time running user processes and 50 percent of its time doing operating system work. This means that rescheduling can only happen 50 percent of the time at most. Measurements show that rescheduling is requested about 10 to 20 percent of the time at most.

In fact, clock interrupts can only make rescheduling requests 1 percent of the time; disk request completions can only make rescheduling requests 33 percent of the time, and terminal i/o completion can only make rescheduling requests 66 percent of the time.

In conclusion, I see the best way to improve ''fairness'' in sharing the central processor is by increasing the number of requests for rescheduling via the clock interrupt. I have experimented in this area. I have tried twice a second and ten times a second. Subjectively, there seems to be better response time when competing with compute bound

jobs. I might suggest that the number of context switches per second depend on the percentage of user time available.

If a reason is necessary for such improvements then consider the following. Rescheduling of the central processor because of device completion is based on the priority of the previous operating system interactions. The clock interrupt provides a convient place to sample priorities of the users and rotate the queue of equal priority processes.

Overhead

Operating system overhead can be usually improved by optimization of code frequently used. Profiling a system is an excellent way to determine which code is frequently executed. Interrupts and system calls are invariably heavily used because they are the start of most transactions that take place in the operating system.

Measurements at University of California at San Francisco showed that system calls take at minimum of 320 microseconds. Thus, the more system calls done; the more overhead incurred. To reduce overhead, system calls should be done as infrequently as possible. Terminal i/o is usually a sore point for the number of system calls.

Interrupts affect system overhead in several ways. Interrupts block out other interrupts at this level and lower. Interrupts steal central processing cycles. To reduce system overhead in this area, the number of interrupts as well as the duration of the interrupts must be reduced. The number of interrupts depends on the mode of transfer and the reason for the interrupts. Terminal i/o is the most frequent interrupt in most systems. The number of interrupts can be reduced by FIFO'ing the data in bursts or using direct memory access devices. A common reason for high system overhead is unterminated lines and noise. Login processes can create a feedback loop with open lines and cause continuous interrupts.

Measurements of terminal i/o shows that 90 percent of all transfers are output and only 10 percent are input. Thus optimization in the area of terminal output may reduce system overhead. One approach to improving character by character output is called ''pseudo dma''. The idea here is to simulate the direct memory access device. In other words, each terminal has a pointer to some storage space and the number of characters at this location. This method provides a quick way to get the character to be outputted and return from interrupt.

Most interrupts can be handled simply and usually do a wake up of some process. The wake up itself becomes the chief expense of the interrupt. Wake ups are expensive be-

cause they do a linear search through the entire process table looking for appropriate processes to be put on the run queue. There are two approaches to reducing the search through tables. These optimization methods can be used through the operating system. One approach is to keep track of the last entry in the table and only search this far. This approach will have a large effect if only a small portion of the table is used.

The second approach is to drop table searches and use queues. Measurements show that most of the time only a single process is waiting on an event. These results support the idea of using queues. I have not heard of or implemented myself this scheme but it might be worth trying.

Another area of system overhead is copying data from user space to kernel space or kernel space to user space. These copies are expensive operations. During these copies interrupts are not allowed. Interrupts can be allowed in this section of code if the supervisor registers are not altered during any interrupt.

There are a number of improvements that can be made to the operating system to reduce overhead. Each improvement, of course, will have a varying amount of reduction of overhead depending on how the system is used.

In closing this section, let me say that a more substantial improvement can be noted in all areas with increased memory speeds even though the relative percentage may not change.

Conclusion

In closing, let me say that the basic principles of good performance have been around for a while. The reason it may seem new is that we have been exploring in new areas and have let past experience in this matter slip away. The only surprizes in this paper may be the models and the figures.

I hope that UNIX installations will now upgrade there performance to match the brilliance of the system underneath.

Acknowledgements

I would like to thank the following people for sharing the results of their measurements or helping me collect information: Professor Fabry, University of California at Berkeley; Bob Kridle, University of California at Berkeley; Tom Ferrin, University of California at San Francisco.

Trademarks

UNIX is a trademark of Bell Laboratories. PDP 11/70, RS04, RP06, RM03, Unibus, and Massbus are trademarks of Digital Equipment Corporation.

Copying Privileges

DRAFT

Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX† Operating System‡

Özalp Babaoğlu

William Joy

Juan Porcar

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

This paper describes a modified version of the UNIX operating system that supports virtual memory through demand paging. The particular implementation being described here is specific to the VAX*-11/780 computer system although most of the design decisions have wider applicability.

The modified system creates a large virtual address space for user programs while supporting the same user level interface as UNIX. The few new system calls that have been introduced are primarily aimed for performance enhancement. The paging system implements a variant of the global CLOCK replacement policy (an approximation of the global *least recently used* algorithm) with a working set like mechanism for the control of multiprogramming level.

Measurement results indicate that the lack of *reference bits* in the VAX memory management hardware can be overcome at relatively little expense through software detection. Also included are measurement results comparing the virtual system performance to the swap based system performance under a script driven load.

Keywords and phrases: UNIX, virtual memory, paging, operating systems, VAX.

December 2, 1979

† UNIX and UNIX/32V are Trademarks of Bell Laboratories

‡ Work supported by the National Science Foundation under grants MCS 7807291, MCS 7824618, MCS 7407644-A03 and by an IBM Graduate Fellowship to the second author.

* VAX and PDP are trademarks of Digital Equipment Corporation.

Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX† Operating System‡

Özalp Babaoğlu

William Joy

Juan Porcar

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

1. Introduction

The most significant architectural enhancement that the VAX-11/780 provides over its predecessor, the PDP*-11, is the very large address space made available to user programs. The fundamental task of transporting UNIX to this new hardware was accomplished by Bell Laboratories at Holmdel. In addition to the portability directed changes, the memory management mechanism of the base system was modified to make partial use of the new hardware. In particular, through these changes, processes could be *scatter loaded* into memory thus avoiding main memory fragmentation, and swapped in and out of memory *partially*. A process, however, still had to be fully loaded in order to execute. While no longer limited by the 16 bit address space of the PDP-11, the per process address space could grow only as large as the physical memory available to user processes. This system, which constituted a prerelease of UNIX/32V†, was adopted as the basis for virtual memory extensions.

The virtual memory effort was motivated by several factors in our research environment:

* To provide a very large virtual address space for user processes, in particular, Lisp systems such as MACSYMA, and other systems employed in Image Processing and VLSI design research.
* To provide an easily accessible virtual memory environment suitable for research in the fields of storage hierarchy performance evaluation and automatic program restructuring.
* To try to improve overall system performance by making better use of our very limited memory resource.

The reader should be familiar with the standard UNIX system as described in [RITC 74] and the virtual memory concept in general [DENN 70]. In the next section, we briefly describe the memory management hardware as it exists in the VAX-11/780 to support virtual memory [DEC 78]. The following sections detail the new kernel operations including new system calls followed by various measurement results.

---

† UNIX and UNIX/32V are Trademarks of Bell Laboratories

‡ Work supported by the National Science Foundation under grants MCS 7807291, MCS 7824618, MCS 7407644-A03 and by an IBM Graduate Fellowship to the second author.

* VAX and PDP are trademarks of Digital Equipment Corporation.

2. VAX-11/780 Memory Management Hardware

The VAX-11/780 memory management hardware supports a two level mapping mechanism to perform the address translation task. The first level page tables reside in system virtual address space and map user page tables. These tables in turn, map the user virtual address space which consists of 512 byte pages. The 32 bit virtual address space of the VAX-11/780 is divided into four equal sized blocks.

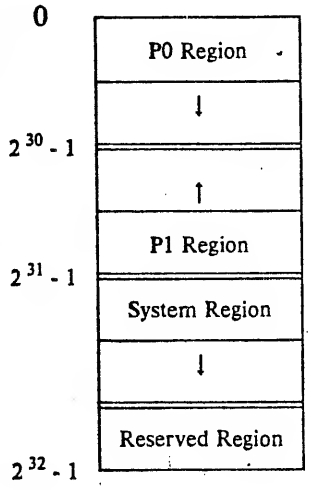


Fig. 2.1. Virtual address space

Two of these blocks, known as the P0 and P1 regions, constitute the two per process segments. The third block, known as the system region, contains the shared kernel virtual address space while the fourth region is not supported by the current hardware. The P0 segment starts at virtual address 0 and can grow toward higher addresses. The P1 segment on the other hand, starts at the top of user virtual address space and grows toward lower addresses. Both segments are described by two per process (base, length) register pairs.

A page table entry consists of four bytes of mapping and protection information. Attempting a translation through a page table entry that has the *valid* bit off results in a *Translation Not Valid Fault* (i.e., a page fault). Whereas most architectures that support virtual memory provide a per page *Reference Bit* that is automatically set by the hardware when the corresponding page is referenced to be examined and/or reset by the page replacement algorithm, the VAX-11/780 has no such mechanism. In order to overcome this deficiency, we distinguish the three states that a page of virtual address space can be in:

[1] Valid — the page is in memory and valid. This corresponds to the *valid, referenced* state in the presence of a reference bit.

[2] Not valid but in memory — the page is in memory but the page table entry is marked not valid so as to cause a page fault upon reference. This is the so called *reclaimable* state of the page. Equivalent to the *valid, not referenced* state.

[3] Not valid and not in memory — the page is in secondary storage. Equivalent to the *not valid* state.

This scheme in effect allows us to detect and record references to pages using software. We discuss the cost and effectiveness of the method in §7.2.

3. Process Structure

In UNIX, the notion of a *process* and a computer execution environment are intimately related [THOM 78]. In fact, a process is the execution of this environment which consists of the process virtual address space state, general register contents, open files, current directory, etc. The state of this pseudo computer is comprised of the contents of four segments. The first three contain the process virtual address space, while the fourth segment describes the system maintained state information.

The process virtual address space consists of a read only program text segment that is shared amongst all processes that are currently executing the same program, as well as private writable data and stack segments. Within the limited segmentation capability of the VAX-11/780, these three segments are mapped such that the program text is in the P0 region beginning at virtual address 0 with the data immediately after it starting at the next page boundary. The stack segment is mapped into the P1 region starting at the highest virtual address. While the text segment has a static size, the data segment can be grown or shrunk through system calls and the stack segment is grown automatically by the kernel upon the detection of segmentation faults.

The physical structure of these segments in secondary storage (called an *image*) can be organized in various ways. At one extreme is the physically contiguous organization described simply by a (base, length) pair. While appropriate for static segments, such as text, this organization is too rigid for dynamically growing segments, like the data and stack segments. In addition to fragmentation, segment growth beyond the current allocation could imply physical movement of the image. At the other extreme is a fully scattered organization of the image. While minimizing fragmentation, this can result in expensive allocation and mapping functions due to the large number of pages which are present in large images.

The image organization chosen for the dynamic segments represents a compromise between the two extremes. Each image consists of several scattered chunks. An exponentially increasing chunk allocation scheme allows the mapping of very large segments through a small table. Limiting the maximum size of any chunk helps to prevent extreme fragmentation. Thus in the current system, the smallest chunk allocated to a segment is 8K bytes, and chunk sizes increase by powers of two up to a maximum size of 2M bytes.

4. Kernel Functions

We now describe the major new functions performed by the kernel as well as the existing functions whose implementation have been significantly modified. For the purposes of future discussion, we define the following terms:

Free List — The doubly linked circular queue of page frames available for allocation. Allocation is always from the head, while insertion occurs both at the head (for pages which can no longer be needed) or the tail (for pages which might still be reclaimed).

Loop — Envision the set of physical page frames that are not in the free list as if they were arranged statically around the circumference of a circle. We refer to these set of page frames, ordered by physical address, as the *loop*. Page frames allocated to kernel code and data appear in neither the loop nor the free list.

Hand — A pointer to a page frame that is in the loop. The hand is incremented circularly around the loop by the pageout daemon as described below.

4.1. Page Fault Handling

The most visible of the kernel changes is the existence of a *Translation Not Valid* fault handler. Given the virtual address that caused the fault, the system checks to see if the page containing the virtual address is in the *reclaimable* state. This happens when the pageout

daemon has swept past a page and made it reclaimable to simulate a reference bit (as described below). If the page is in this state, it can once again be made valid, and the process returns to user mode. Note that if the reclaimed page was in the free list, it is removed and reenters the loop. Since none of the operations involved in reclaiming a page can cause the process to *block*, reclaiming a page does not involve a processor context switch and reschedule.

If the page cannot be reclaimed (i.e., is not no longer in core), then a page frame is allocated and the disk transfer is initiated from the segment image as dictated by the image mapping.

In reality, more cases must be considered. If the faulting page belongs to a shared text segment, the disk transfer is initiated only if the page is not reclaimable and not *intransit*, i.e., the pagein operation has not already been initiated by another process that is sharing the text segment. If intransit, the faulting process sleeps to be awoken by the process that started the page transfer when it completes. Here we note that the first level page tables for shared text segments are *not* shared, but rather, each process has its own copy.† Thus, all operations that modify page table entries of shared text segments must insure that all existing copies are updated.

Other types of page faults that require special handling are those where the faulting page is marked as *fill on demand*. There are three types of demand fill pages:

Zero Fill These pages are created due to segment growth and result in a page of zeroes when referenced.

Text Fill These pages result from execution of demand-loaded programs, and cause the corresponding page to be loaded at the point of first reference, from the currently executing object file. Such object files are created by a special directive to the loader and are described further in §5.3.

File Fill These pages are similar to text fill pages, but the pages come from a open file rather than the current text image file. These pages are set up by the **vread** system call. See section §5.2 for more details.

4.2. Page Write Back

During system initialization, just before the *init* process is created, the bootstrapping code creates process 2 which is known as the *pageout daemon*. It is this process that actually implements the page replacement policy as well as writing back modified pages. The process leaves its normal dormant state upon being woken up due to the memory free list size dropping below an upper threshold.

At this point, the daemon examines the page frame being pointed to by the *hand*. If the page frame corresponds to a valid page, it is made reclaimable. Otherwise the page was reclaimable, and it is freed, but remains reclaimable until it is removed from the free list and allocated to another purpose. The hand is then incremented and the above steps are repeated until either the free memory is above the upper threshold or the angular velocity of the hand exceeds a bound.

The rate at which the daemon examines page frames increases linearly between the free memory upper threshold and lower threshold (these are tunable system parameters). In a loaded system the hand will be moved around the loop two to three times a minute.

Upon encountering a reclaimable page that has been modified since it was last paged in, the daemon must arrange for it to be written back before the page frame containing it can be freed. To accomplish this, the daemon queues a descriptor of the I/O operation for the paging device driver. Upon completion of the I/O, the interrupt service routine inserts the descriptor

† Sharing all user level page tables of shared segments would require a 64K byte alignment between the text and data segments. This is not enforced by the current loading scheme, so currently these page tables are not shared at all.

to the *cleaned list* for further processing by the daemon. The daemon periodically empties the cleaned list by freeing the page frames on it that have not been reclaimed in the meantime.

Note that as described above, this pageout process implements a variant of the global CLOCK page replacement algorithm that is known as *scheduled sweep* [CORB 68, EAST 79].

4.3. Process Creation

In UNIX, every process comes into being through a **fork** system call whereby a copy of the *parent* process is created and identified as the *child*. This involves the duplication of the parent's private segments (data and stack) and the system maintained state information (open files, current directory, etc.).

Within a virtual memory environment including the pagein and pageout primitives described above, the implementation of the **fork** system call is conceptually very simple. The parent process copies its virtual address space to the child's one page at a time. Note that this may require faulting in the invalid portions of the parent's address space. Since the VAX-11/780 memory management mechanism can establish only one mapping at a time, the child's address space is actually created indirectly through the kernel virtual memory.

Although conceptually simple, the above scheme has undesirable system performance consequences. Duplication of the parent's private segments generates a sharp and atypical consumption of memory. Since a significant percentage of all forks serve only to create system contexts to be passed to another process via the **exec** system call, the copying of the parent's private segments is largely unnecessary. The **vfork** system call, described in §5.1, has been introduced to provide an efficient way to create new system contexts within the current design.

4.4. Program Execution

The **exec** system call, whereby a process overlays its address space also has a simple implementation. The process releases its current virtual memory resources and allocates new ones as determined by the program being executed. Then, the program object file is simply read into the process address space which has been initialized as *zero fill on demand* pages so as not to generate irrelevant paging from the process' old image.

This implementation suffers from the same problems as the above fork implementation. Initiation of very large programs is very slow, and results in system wide performance degradation due to the loading of the entire program file in the virtual memory before execution commences. A new loader format which allows demand paging from the program object file has been designed to improve large program start up and to eliminate this non-demand situation (see §5.3).

4.5. Process Swapping

Swapping a process out involves releasing the physical memory currently allocated to it (called the *resident set*) and writing back its modified pages to its image along with the system maintained state information and page tables. Swapping a process in, on the other hand, involves reading in its page tables and state information and resuming it. Note that as no pages from the process address space are brought in, the process will have to fault them back in as required. The alternative of swapping the resident set in and out is not implemented.

4.6. Swap Scheduling

When the amount of available free memory in the system cannot be maintained at a minimal number of free pages by the pageout daemon, then the system invokes the swap scheduler. In order to free memory, the swap scheduler will select a process which is resident and swap it (completely) out. The scheduler prefers first to swap out processes which have been blocked for a significant length of time, and chooses the process which has been in such a state the longest. If there are no such processes, and it is therefore necessary to swap out a process which is or has recently been active, the system chooses from among the remaining

processes the one which has been memory resident the longest.

In choosing an active process to swap out, the system checks to guarantee that the process has had a minimal amount of time necessary to demand fault in the number of pages which it had wh n it was last swapped out (based on maximum expected paging device throughput). This ser es to guarantee a minimal amount of progress by a process each time it is swapped in. When a process is forced out while it has many pages, it is given lower priority to return to the set of resident processes than ones which swapped with fewer pages or which are very small.

The swap-in mechanism also recognizes that processes which swapped out with many pages, will need to fault in pages when they are brought in. The system therefore maintains a notion of a global memory *deficit,* which is the expected short term demand for memory from processes recently brought in, based on the number of pages they were using when they swapped out. The deficit is charged against the free memory available when deciding whether to bring a process in.

In general, this swap scheduling mechanism does not perform well under very heavy load. The system performs much better when memory partitioning can be done by the page replacement algorithm rather than the swap algorithm. If heavy swapping is to occur on moving head devices, then better algorithms could be implemented. High speed specialized paging devices, on the other hand, would suggest different algorithms based on migration.

4.7. Raw I/O

In a virtual memory environment, handling input/output operations directly to/from process address space without going through the system buffer cache requires special attention. The pages involved in the I/O must be insured to be valid and locked for the duration of the operation. This is accomplished through the *virtual segment lock/unlock* internal primitives. Locking a virtual segment consists of locking pages that are already valid and faulting/reclaiming invalid pages by simply touching them and refraining from unlocking the page frame (which is allocated in the locked state) after the pagein completion.

5. New System Facilities

5.1. Process Creation

In order to allow efficient creation of new processes, a new system primitive **vfork** has been implemented. After a **vfork**, the parent and its virtual memory run in the child's system context, which may be manipulated as desired for the new image to be created. When a **exec** is executed in the child's context, the virtual memory and parent thread of control are returned to the parent's context and a new thread of control and virtual memory are created for the child. This mechanism allows a new context to be created without any copying.

It should be noted that an implementation of **fork** using a *Copy On Write* mechanism would be completely transparent and nearly as efficient as **vfork.** Such a mechanism would rely on more general sharing primitives and data structures than are present in the current version of the system, so it has not been implemented.

5.2. Virtual Reading/Writing of Files

In order that efficient random access be permitted in a portable way to large data files, a pair of new system calls has been added: **vread** and **vwrite.** These calls resemble the normal UNIX **read** and **write** calls, but are potentially much more efficient for sparse and random access to large data files. **Vread** does not cause all the data which is virtually read to be immediately transferred to the users address space. Rather, the data can be fetched as required by references, at the systems discretion. At the point of the **vread,** the system merely computes the disk block numbers of the corresponding pages and stores these in the page tables. Faulting in a page from the file system is thus no more expensive than faulting in a page from the swap

device. In both cases all the mapping information is immediately available or can be easily computed from incore information. **Vwrite** works with **vread** to allow efficient updating of large data which is only partially accessed, by rewriting to the file only those pages which have been modified.

Downward compatibility wit non-virtual systems is achieved by the fact that **read** and **write** calls have the same semant s as **vread** and **vwrite** calls; only the efficiency is different. Upward extensibility into a more general sharing scheme is also easy to provide, as **vread** can be easily simulated by a mapping of the file into the address space with a copy-on-write mechanism on the pages. Although the current mechanism does not share copies of the same page if it is **vread** twice, the semantics of the system call do not prohibit such an implementation if used with a copy-on-write mechanism. Note that **vwrite** can also be simulated by a map-out like mechanism.

5.3. New Loader Format

The same mechanism that is used to implement the **vread** system call is used to provide a load format where the pages of the executable image are not pre-loaded, but rather initialized on demand, with the page block numbers only being bound into the page tables at the point of **exec.** The only change from the other UNIX load formats in this new format is the alignment of data in the load file on a page boundary. Large processes using this format can begin execution very quickly, with page faults causing reading from the executed file.

6. Perspective

There are a number of facilities which have not been implemented in the first release of the system as described here.

For example, there are plans to change the system to use 1024 byte disk blocks rather than 512 byte blocks. It has been observed that in many cases the system is limited by the number of disk transactions that can be made per second. Larger disk blocks will help improve disk throughput. On machines with large real memories, using page-pairs in the paging system will also reduce the overhead of the replacement algorithm and increase throughput to the paging device. Since a page contains only 128 words, it does not provide a great deal of locality. It is expected that using 1024 byte pages (in effect) will not degrade the effective memory size significantly.

Another problem associated with the small page size of the VAX architecture is the rate of growth of user page tables.† For very large processes, this not only results in a significant amount of real memory allocated to page tables, but also increases the system overhead in dealing with them. Effectively supporting extremely large virtual address spaces will require handling translation faults at the first level (i.e., page table faults) whereby real memory for page tables is allocated on demand.

The system changes as presented here are the result of approximately one man year of effort. The base system (a prerelease of UNIX/32V that was maintained as the production system during the paging development) and the paged system consist of approximately 14800 and 16800 total source lines, respectively. The brake down of these numbers amongst the various types of source is presented in Table 6.1. Also presented is a comparison that excludes comment lines from the source of the two systems.‡ We note that the actual number of lines *modified* to obtain the paged system is considerably more than the simple net difference for each category. In the meantime, for equal configurations, the resident kernel size has increased by about 12K bytes of program and 26K bytes of data resulting in a total size of about 164K bytes (for a 1 megabyte main memory system).

† Since a page table entry is 4 bytes, user page tables grow one byte for each 128 bytes of user virtual memory.

‡ For the C source code, the number of occurrences of ";" was used as an estimate of the actual number of source lines that were not comments.

Category	Total Source Lines		Non Comment Lines	
	Base System	Paged System	Base System	Paged System
Assembly Code	1292	1353	1062	1015
C Code	11581	13405	4891	5614
Header Files	1997	2068	1223	1316

Table 6.1. Source Code Volume Comparison

7. Measurement Results

The system has been instrumented to collect data related to various paging system activities as well as workload characteristics in general.

7.1. Process Virtual Size Distribution

Being one of the few quantifiable characteristics of a workload that is also of importance in a virtual memory environment, system-wide distribution of process virtual size was monitored.

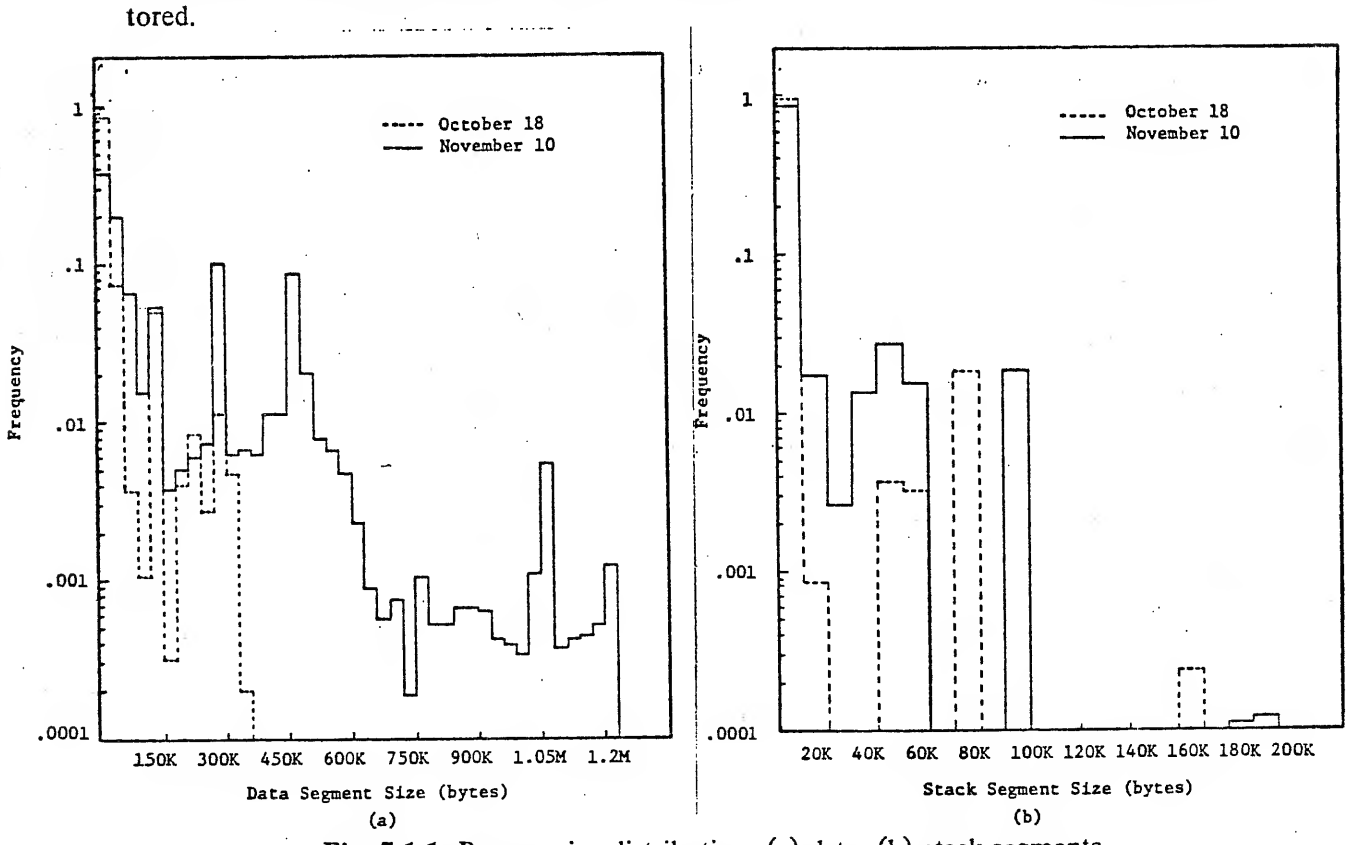


Fig. 7.1.1. Process size distribution: (a) data, (b) stack segments

The results of integrating process data and stack segment size over user CPU time are shown in Fig. 7.1.1. The two sets of measurements taken almost one month apart indicate an increase from 29.6K to 161.7K bytes and 6.8K to 9.8K bytes for the mean data and stack segment sizes, respectively. The October 18 measurement corresponds to early stages of production run of the system and is representative of the pre-virtual memory workload. The significant increase in the

average data segment size within this short period is attributed to the rapid growth of Lisp systems including MACSYMA. The insignificant contribution of the stack segment to total process size is a characteristic of our C intensive workload.

7.2. Page Fault Service Time Distribution

As described in §2, a page can be in three states. Reference to a page in memory but invalid causes a *reclaim*, whereas reference to one not in memory results in a *page-in* operation. The service time distributions for these two different types of faults is shown in Fig. 7.2.1.

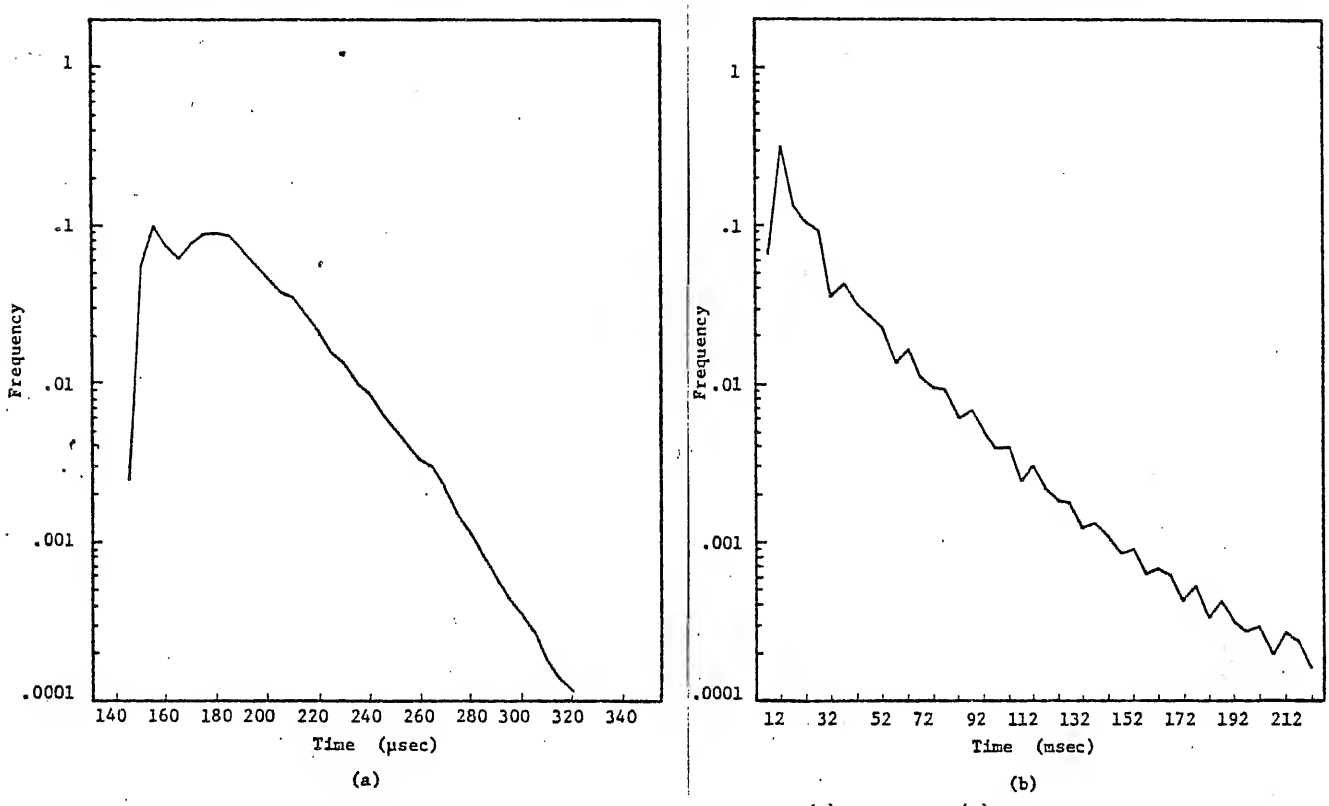


Fig. 7.2.1. Fault service time distribution: (a) reclaim, (b) page-in

For the reclaim time distribution, the first peak corresponds to reclaims from the *loop*, while the second bump and the long tail are accounted for by the load dependent component of the service time due to reclaiming pages of shared text segments.† Note that the mean reclaim time of 208 microseconds per reclaim represents a negligible delay to user programs. Furthermore, the overall system cost of reclaims through which we simulate the missing reference bits of the architecture has been measured to be less than 0.05% of all CPU cycles.‡

The page-in service time distribution is highly load dependent since it includes all of the queueing as well as process rescheduling delays. The configuration with the paging activity on the same arm (an RM-03 equivalent disk) as the temporary and the root file systems results in a 54.9 msec total service time. The significant number services completed under 20 msec are due to the track buffering capability of the controller being used.

† This operation requires updating the page tables of all processes currently executing the same code, thus varies with load.

‡ This cost is actually a function of the paging activity. The number reported here has been averaged over a 28 hour period in a 1.25M byte real memory configuration

7.3. Comparison with Swap Based System

In an effort to compare the performance of the system before and after the addition of virtual memory, a script driven workload was run in a stand-alone manner in both systems under identical configurations consisting of a 1 megabyte main memory, an RP-06 servicing the user file system and an RM-03 shared by the root and temporary file systems in addition to the swapping/paging activity. The swap based system used for this comparison was quite sophisticated, performing scatter loading of processes into memory and partially swapping processes to obtain free memory.

The basic unit of work generated by the script is made up of four concurrent terminal sessions:

lisp A recompilation, using a Lisp compiler, of a portion of the compiler, and a "dumplisp" using the lisp interpreter to create a new binary version of the compiler. Under the paging system, a system load format which caused the interpreter and compiler to be demand loaded (rather than preloaded) was used (cf. §5.3).

ccomp An editor session followed by a recompilation and loading of several C programs which support the line printer.

typeset An editor session followed by typesetting of a paper involving mathematical processing, producing output for a Versatec raster plotter.

trivial Repeated execution of a trivial command (printing the date) every few seconds.

Staggered multiple initiations of from one to four of these terminal sessions were used to create increasing levels of load on the system. Figure 7.3.1 gives the average completion time for each category of session under the two systems.

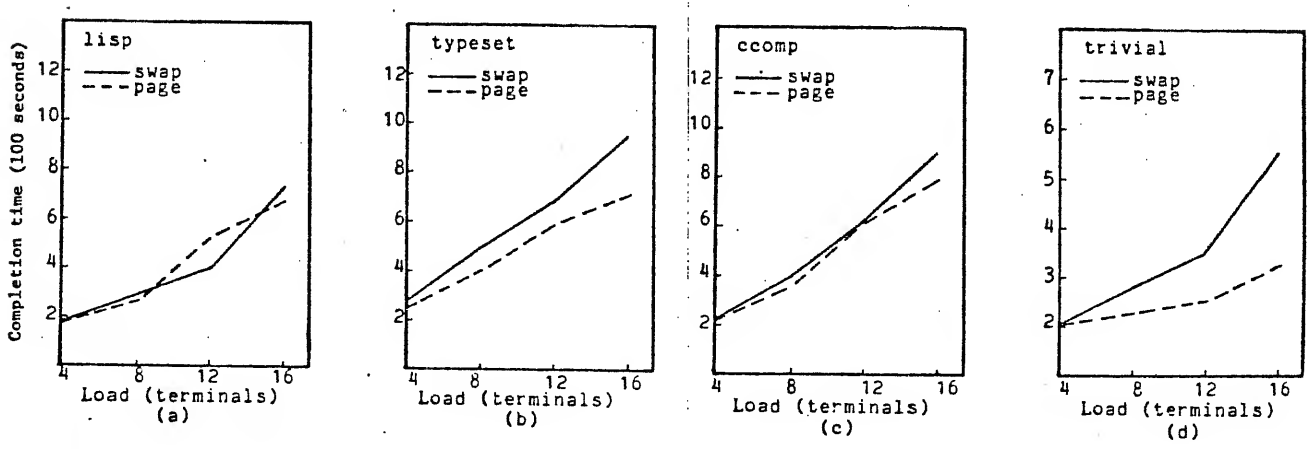


Fig. 7.3.1. Average completion times
(a) lisp, (b) typeset, (c) ccomp, (d) trivial

For the non-trivial sessions, completion times were very similar under the two systems, with the paging version of the system running (in all but one case) faster, and the swap based system departing from linear degradation more rapidly. This trend is most noticeable in the response time for the trivial sessions. Systemwide measures collected during the experiments are given in Figure 7.3.2.

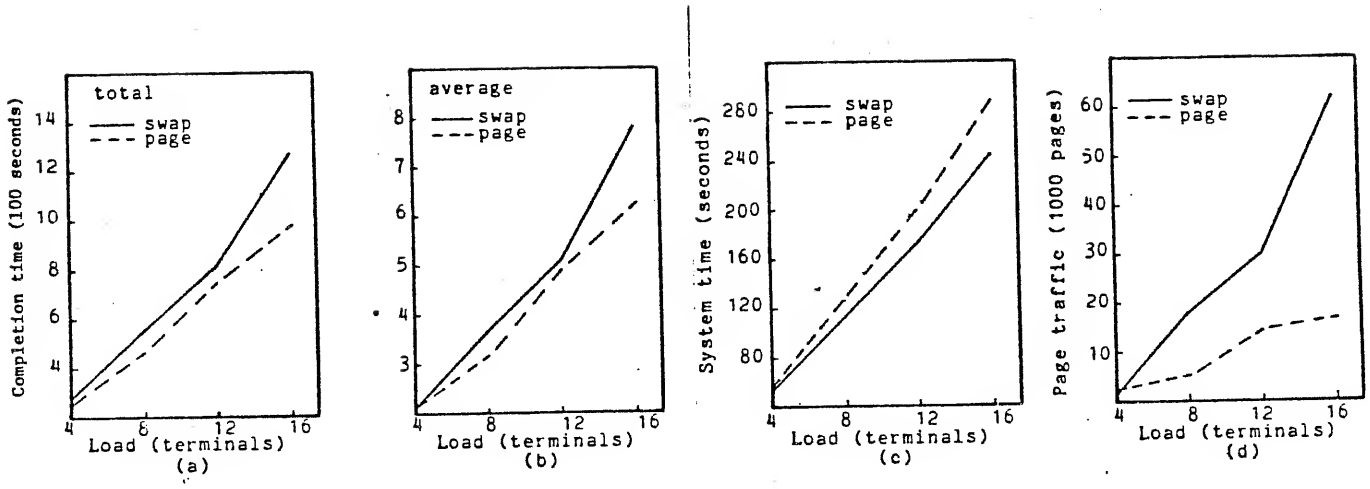


Fig. 7.3.2. Systemwide measurements
(a) total (b) average completion time, (c) system time, (d) total page traffic

These measurements show the same trend for both total and average completion times as for individual sessions, with the paging system slightly faster and degrading more linearly within the measured range. System overhead was uniformly greater under the paging system, constituting 26 percent of the CPU utilization as compared to 20 percent under the swap system. User CPU utilization was, however, uniformly greater under the paging system, averaging 48 percent, while the swap based system averaged only 42 percent.

Finally, the total page traffic generated by the two systems was measured. This accounts for both paging and swapping traffic under the paging system, as well as transfer of all system information (control blocks and page tables) under both systems. Although the paging system resulted in far fewer total pages transferred, the actual number of transactions required to accomplish this was much greater since most data transfers, under the paging system, are due to paging rather than swapping activity, and thus take place in very small (512 byte) quantities. We are currently installing modifications in the system to use larger block sizes in both the file and paging subsystems, and expect improved performance from these changes.

Acknowledgements. The cooperation of Bell Laboratories in providing us with an early version of UNIX/32V is greatly appreciated. We would especially like to thank Dr. Charles Roberts of Bell Laboratories for helping us obtain this release, and acknowledge T. B. London and J. F. Reiser for their continuing advice and support.

We are grateful to Domenico Ferrari, Richard Fateman, Jehan-François Pâris, William Rowan, Keith Sklower and Robert Kridle for their participation in the early stages of the design project, and would like to thank our user community for their patience during the system development period.

References

[CORB 68] F. J. Corbato, "A Paging Experiment with the Multics System," Project MAC Memo MAC-M-384, July, 1968, Mass. Inst. of Tech., published in *In Honor of P. M. Morse*, ed. Ingard, MIT Press, 1969, pp. 217-228.

[DEC 78] *VAX-11/780 Hardware Handbook*, Digital Equipment Corporation, 1978.

[DENN 70] P. J. Denning, "Virtual Memory," *Computer Surveys*, vol. 2, no. 3 (Sept. 1970), pp. 937-944.

[EAST 79] M. C. Easton and P. A. Franaszek, "Use Bit Scanning in Replacement Decisions," *IEEE Trans. Comp.*, vol. 28, no.. 2 (Feb. 1979), pp. 133-141.

[RITC 74] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Commun. Assn. Comp. Mach.*, vol. 17, no. 7 (July 1974), pp. 365-375.

[THOM 78] K. Thompson, "UNIX Implementation," *Bell System Tech. Journal*, vol. 57, no. 6 (July-Aug. 1978), pp. 1931-1946.

Getting started with...

Berkeley Software for UNIX† on the VAX‡

(The third Berkeley Software Distribution

A package of software for UNIX developed at the Computer Science Division of the University of California at Berkeley is installed on our system. This package includes a new version of the operating system kernel which supports a virtual memory, demand-paged environment. While the kernel change should be transparent to most programs, there are some things you should know if you plan to run large programs to optimize their performance in a virtual memory environment. There are also a number of other new programs which are now installed on our machine; the more important of these are described below.

Documentation

The new software is described in two new volumes of documentation. The first is a new version of volume 1 of the UNIX programmers manual which has integrated manual pages for the distributed software and incorporates changes to the system made while introducing the virtual memory facility. The second volume of documentation is numbered volume 2c, of which this paper is a section. This volume contains papers about programs which are in the distribution package.

Where are the programs?

Most new programs from Berkeley reside in the directory **/usr/ucb**. A major exception is the C shell, *csh,* which lives in **/bin**. We will later describe how you can arrange for the programs in the distribution to be part of your normal working environment.

Making use of the Virtual Memory

With a virtual memory system, it is no longer necessary for running programs to be fully resident in-memory. Programs need have only a subset of their address space resident in memory, and pages which are not resident in memory will be *faulted* into memory if they are needed. This allows programs larger than memory to run, but also places a penalty on programs which do not exhibit *locality*. It is important to structure large programs so that at any one time they are referring to as small a number of pages as possible.

If you are going to create very large programs, then you should know about a new demand load format. This format causes large programs to begin execution more rapidly, without loading in all the pages of the program before execution begins. It is most suitable for programs which are large (say > 50K bytes of program and initialized data), and especially when a program has a large number of facilities, not all of which are used in any one run. To create an executable file with this format, you use the **−z** loader directive; thus you can say "cc −z ..." or "f77 −z ...". The *file* command will show such files to be "demand paged pure executable" files. See the manual page for *ld* in section 1 and *a.out* in section 5 of volume 1 of the manual for more information.

If you have or are writing a large program which creates new processes as children, then you should know about **vfork** system call. The **fork** system call creates a new process by copying the data space of the parent process to create a child process. In a virtual environment this is very expensive. **Vfork** allows creation of a new process without copying the parent's address

†UNIX is a trademark of Bell Laboratories.
‡VAX is a trademark of Digital Equipment Corporation.

space by letting the parent execute in the child's system context. The parent can set up the input/output for the child and then return to its own context after a call to exec or _exec. If you use the standard I/O routine **system()** to execute commands from within your programs, then **vfork** will be used automatically. If you have been calling **fork** yourself, you should read the manual page for **vfork** and use it when you can.

In order that efficient random access be permitted in a portable way to large data files, a pair of new system calls has been added: **vread** and **vwrite**. These calls resemble the normal UNIX **read** and **write** calls, but are potentially much more efficient for sparse and random access to large data files. **Vread** does not cause all the data which is virtually read to be immediately transferred to your address space. Rather, the data can be fetched as required by references, at the systems discretion. At the point of the **vread**, the system merely computes the disk block numbers of the corresponding pages and stores these in the page tables. Faulting in a page from the file system is thus no more expensive than faulting in a page from the paging device. In both cases all the mapping information is immediately available or can be easily computed from incore information. **Vwrite** works with **vread** to allow efficient updating of large data which is only partially accessed, by rewriting to the file only those pages which have been modified.

Downward compatibility to non-virtual systems is achieved by the fact that **read** and **write** calls have the same semantics as **vread** and **vwrite** calls; only the efficiency is different. If you have programs which access large files, and do so sparsely, read the manual pages for **vread** and **vwrite** in section 2 of volume 1 of the manual.

New Languages for the VAX

There are now available interpreters for APL and Pascal for the VAX, and a LISP system supporting a dialect of LISP compatible with a large subset of MACLISP. The APL interpreter is the 11 version, moved to the VAX, and now has a large workspace capability (but has not been extensively used.) The Pascal system has been used extensively for instruction and research and is the same system which was available on the PDP-11. The only limitations of the Pascal system are a maximum of 32K bytes per stack frame (due to the implementation of the interpreter), and 64K bytes per variable allocated with *new*. Essentially arbitrary sized programs can be run with the system, which supports a very standard Pascal with no language extensions. The Pascal system features very good error diagnostics, and includes a source level execution profiling facility.†

The LISP system, "Franz Lisp", was developed at Berkeley as part of a project to move the MIT MACSYMA system from the PDP-10 to the VAX. A compiler *liszt* for Franz Lisp, written at Bell Laboratories, is also included with the system.

For more information about APL refer to its manual page in volume 1 of the manual. The Pascal system consists of the programs **pi, px, pix, pxp, pxref,** and **pic,** all of which are documented in section 1 of volume 1 of the manual. There is also a paper introducing the system in volume 2c. The LISP system is described in *The Franz Lisp Manual* in volume 2c of the manual.

A display editor — *vi*

The system includes the latest version of the display editor *vi* which runs on a large number of intelligent and unintelligent display terminals. This editor runs using a terminal description data base and a library of routines for writing terminal independent programs which is also supplied. The editor has a mnemonic command set which is easy to learn and remember, and deals with the hierarchical structure of documents in a natural way. Editor users are protected against loss of work if the system crashes, and against casual mistakes by a general *undo* facility as well as visual feedback. The editor is quite usable even on low speed

† A compiler for Pascal based on this system is currently being developed, but is not part of this distribution.

lines and dumb terminals.

For users who prefer line oriented editing, the *ex* command enters the same editor, but in a line oriented editing mode. For beginners who have never used a line editor before, there is a version of the editor known as *edit* which has a well-written tutorial introducing it.

For more information about *edit* see *Edit: a Tutorial* in volume 2c of the manual. The line editor features are described in the *Ex Reference Manual* which is in volume 2c of the manual. Also in volume 2c are *An Introduction to Display Editing with Vi* and a *vi* reference card.

Command and mail processing programs

There is also a new command processor *csh* which caters to interactive users by providing a history list of recent commands, which can be easily repeated. The shell also has a powerful macro-like aliasing facility which can be used to tailor a friendly command environment. *Csh* is implemented so that both it and the standard shell **/bin/sh** can be run on the same system.

The *Introduction to the C shell* introduces the shell. If you have used the standard shell, then you should especially read about the *history* and *alias* mechanisms of the shell.

In order that the manual distributed with the tape correspond to the commands which are available to you, you should set the execution search path in your shell startup file to include **/usr/ucb**. If you use **/bin/sh** you should place the lines:

```
PATH=:/usr/ucb:/bin:/usr/bin
export PATH
```

in the file **.profile** in your home directory. If you use *csh*, then the commands

```
setenv PATH /usr/ucb:/bin:/usr/bin:
set path=(/usr/ucb /bin /usr/bin .)
```

should be placed in your file **.cshrc**, also in your home directory. The C shell hashes the locations of commands so as to speed command execution. Placing the current directory "." at the end of your search path speeds the location and execution of commands.

For sending and receiving mail, a new interactive mail processing command provides a hospitable environment, supporting items such as subject and carbon copy fields, and allowing creation of distribution lists. This command also has a mail reading mode which makes it convenient to deal with large volumes of mail. See the manual page for *mail* in section 1, volume 1 of the manual, and the *Mail reference Manual* in volume 2c of the manual for more details.

Better debugger support

A version of the symbolic debugger *sdb* is provided which now can debug FORTRAN 77 programs. The assembler has been rewritten and the C compiler modified to reduce greatly the overhead of using the symbolic debugger, making it much more feasible for heavy use. If you are interested, then you should read the new document for *sdb*, provided in volume 2c.

Other software

The distribution includes a copy of the ciruit analysis program SPICE, which has been transported to the VAX. You can read about it in its reference manual in volume 2c of the manual. Other new programs include programs to simulate the phototypesetter on 200 bpi plotters, a common system messages facility, routines for data compression, a modified version of the standard I/O library permitting simultaneous reads and writes, a network for connecting heterogeneous UNIX systems at low cost (1 tty port per connection per machine and no system changes), and a new, flexible macro package for n/troff *−me*. New command **whatis** and **apropos** can be used to identify programs and to locate commands based on keywords. Try

```
cd /usr/ucb
whatis *
```

and to find out about Pascal:

```
apropos pascal
```

Monitoring the new system

If you want to see what is happening in the new system, you can use the new **vmstat** command, described in section 1 of the manual, which shows the current virtual load on the system. The system recomputes the information printed by **vmstat** every five seconds, so a "vmstat 5" is a good command to try.

To see what processes are active virtual processes, you can do

```
ps av
```

The command

```
ps v
```

will print only the active processes which you are running.

Berkeley Software for UNIX† on the VAX‡

(The Third Berkeley Software Distribution)

A new package of software for UNIX will be available from the Computer Science Division of the University of California at Berkeley in early December, 1979. This is a package of software for UNIX/32V† licensees, and includes a paged version of the kernel for the VAX, as well as a large number of other programs. The tape includes:

New Languages for the VAX

Interpreters for APL, LISP and Pascal. The APL interpreter is the PDP-11 version, moved to the VAX. The LISP system, known as "Franz Lisp", is written in C and LISP, includes both an interpreter and a compiler, and is compatible with a large subset of MACLISP. The Pascal system is the instructional system which has been distributed previously for PDP−11's‡. The language implemented is very close to standard Pascal, and features excellent diagnostics, and a source level execution profiling facility.

New System Facilities

The system is now fully and transparently demand paged. As distributed it will support individual process sizes up to 8M of data and 4M of program. These numbers can be increased to 16M bytes of data and stack and 16M bytes of program easily given the availability of a reasonable amount of disk space to which to page. Description is given of steps necessary to further increase these limits.

A new load-on-demand format allows large processes to start quickly. A *vfork* system call allows a large process to execute other processes without copying its data space. Virtual versions of the *read* and *write* system calls known as *vread* and *vwrite* permit fast random access to large files, fetching data pages as needed, and rewriting only changed pages. The system supports UNIBUS disk drives, and can access and update files on the console's floppy disk drive.

A display editor

The tape includes the display editor, *vi*, (vee-eye) which runs on a large number of intelligent and unintelligent display terminals. This editor uses a terminal description data base and a library of routines for writing terminal independent programs which is also supplied. The editor has a mnemonic command set which is easy to learn and remember, and deals with the hierarchical structure of documents in a natural way. Editor users are protected against loss of work if the system crashes, and against casual mistakes by a general *undo* facility as well as visual feedback. The editor is quite usable even on low speed lines and dumb terminals.

Command and mail processing programs

The tape also includes a new command processor *csh* which caters to interactive users by providing a history mechanism so that recently given commands can be easily repeated. The shell also has a powerful macro-like aliasing facility which can be used to tailor a friendly, personalized, command environment. A new interactive mail processing command supports items such as subject and carbon copy fields, and distribution lists, and makes it convenient to deal with large volumes of mail.

---

†UNIX and UNIX/32V are trademarks of Bell Laboratories.
‡VAX and PDP are trademarks of Digital Equipment Corporation.

Better debugger support

A version of the symbolic debugger *sdb* is provided which now can be used to debug FORTRAN 77 programs. The assembler has been rewritten and the C compiler modified to reduce greatly the overhead of using the symbolic debugger.

Other software

Also included are a number of other useful packages including the circuit analysis program SPICE, programs to simulate the phototypesetter on 200 bpi dot-matrix plotters (these programs were moved from the PDP−11 to the VAX and a large number of fonts available on the ARPANET have been converted to the required format), a bulletin board program, routines for data compression, a modified version of the standard I/O library permitting simultaneous reads and writes, a slow-speed network for connecting heterogeneous UNIX systems at low cost (1 tty port per connection per machine and no system changes), and a new, flexible macro package for *nroff* and *troff* called *−me*.

Source code, binaries and machine readable versions of all documentation are included with the tape. We supply the magnetic tape on which the software is written.

To receive the tape make two additional copies of the the attached agreement, sign and return 2 of the 3 copies with a **check** for $200 U.S. payable to "Regents, University of California," and a copy of your UNIX/32V license agreement to:

Berkeley Software Distribution for UNIX
c/o Keith Sklower
Computer Science Division, Department of EECS
Evans Hall
University of California, Berkeley
Berkeley, California 94720

We will return a fully executed copy of the agreement to you with the distribution.

Included with the tape will be two volumes of documentation. The first is a programmers manual (similar to UNIX/32V Volume 1) modified and updated to correspond accurately to the distributed system. The second is a volume of documents (Volume 2C) similar to the two standard volumes (2A and 2B) describing the major packages on the tape.

If you have questions about this tape they can be directed to Keith Sklower at the address above or at (415) 642-4972 or leave messages for Keith at 642-1024.

Comments on the performance of UNIX† on the VAX‡

William Joy

Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, Ca. 94720

Introduction

A recent paper written by David Kashtan of SRI discussed some measurements he made comparing the performance of the two available operating systems for the VAX: UNIX and VMS. The UNIX system measured by Kashtan was UNIX/32V as extended by U.C. Berkeley to support virtual memory. The measurements were made on version 2.1 of the Berkeley system and version 1.6 of VMS. This note seeks to help interpret these results and more clearly point out the differences in approach and current state of the two systems.

We will show that the differences that Kashtan measured have simple explanations, and that the comparison of performance on such benchmarks can guide short-term tuning. Long-term decisions as to which system to run have been made on the basis of portability and flexibility considerations. The results reporte here confirm the correctness of the choice of UNIX on these grounds by confirming its flexibility. The detected slownesses in UNIX are neither fundamental to the way UNIX does business, nor difficult to mitigate if it is felt important to work on the areas in question. We will discuss simple changes to the UNIX system that have been made improving performance in the areas mentioned. The improvements described here were incorporated in the production system at Berkeley during the first three weeks of March, 1980.

System call overhead

Let us first consider system call overhead, which underlies the measurements that were made of "Context Switching Performance." From the measurements Kashtan gives for context switching overheads, we can first estimate some fundamental times for the VMS system: a simple system call (e.g. an event flag call) on VMS appears to take about 114μsec, and a context switch about 178μsec. To get an idea of scale here, the VAX memory cycle time (for cache write-through) is about 1.2μsec, the simplest procedure call **jsb** and matching return **rsb** takes about 4μsec, and the high level language call instruction **calls** and matching return **ret** takes a minimum of 16μsec, more if some registers are to be saved.

On the version of UNIX Kashtan had, a trivial system call took about 350μsec. This is almost three times as much time as VMS used. Where was this time being spent? The following is a rough breakdown:

† UNIX is a Trademark of Bell Laboratories.
‡ VAX and VMS are trademarks of Digital Equipment Corporation.

save registers, call *trap*	40μsec
setup in *trap* for syscall	30μsec
fetching arguments for syscall	60μsec
saving context in case interrupted	100μsec
system call proper	30μsec
checking to see if signals waiting	30μsec
recomputing process priority	60μsec
restore registers, **rei**	30μsec

This code is all modularly and straightforwardly written, using a small set of primitives (for instance the "non-local-goto" at the point of an interrupted system call is effected by using part of the primitives performing context switching.) There are a many calls to small routines here. Each system call argument is fetched by calling a primitive *fuword* which fetches a word from user-space after performing access checks. The saving of the context involves a procedure call, as does checking for signals and recomputing priority. Altogether, when a system call has two arguments, seven **calls** instructions are performed here to various subroutines. These alone take roughly 105μsec, assuming no registers are specified to be saved in the entry mask. Since the saving and restoring of the registers for the call to *trap* takes an additional 20μsec, this means that 125μsec is accounted for without any of the work for the system call itself. Clearly, if this is to be sped up, some of these routine calls must be eliminated.

With minor changes to the code for trap handling we have sped up the system call time so that the basic overhead is 140μsec, instead of 350μsec. This was done as follows:

1. The *trap* routine's "entry mask" as generated by the C compiler is modified at boot time to save all possible registers, to avoid saving some registers twice.†
2. The routine to handle system calls was broken out from the handler for all other traps and called *syscall*. This minimizes the changes to the code for handling other traps in making the following improvements, and allows some small simplifications.
3. Fetching of all arguments can be done by a single routine *copyin*, instead of calling *fuword* for each argument.‡ Since the number of argument words to system calls is very small it is easy to expand the *copyin* primitive in line in this critical path. It can be implemented, in this special case, by two basic VAX instructions: a **prober** to determine accessibility, and a **movc3** to move the arguments into system space.*
4. To restore after an interrupted system call it suffices to be able to locate the frame pointer and stack pointer of the original *syscall* procedure. We can effect this context save using just three **movl** instructions.
5. A subroutine call to check if signals are pending to this process can be avoided in almost all cases by first checking the mask of pending signals. This partial inline expansion of the *issig* routine reduces the overhead here by at least 16μsec.
6. Recomputation of the process priority at each system call can be avoided by "unloading" the **p_pri** per-process information field. The system used a single field to encode both the processes user-CPU usage dependent scheduling priority, and priorities related to process blocking during system calls. By adding a **p_usrpri** information field reflecting user-CPU usage, the code in *trap* reduces to an assignment of this priority to the **p_pri** field, instead of recomputing the user priority after each system call. The space overhead is one byte per

† Previously, the assembly language for the system saved all the registers, and made no assumptions about what *trap* itself did, resulting in registers which were used for register variables within *trap* being saved twice.

‡ The copy of arguments into system space is important to avoid severe system security problems with system calls that self-modify their arguments after they have been checked for consistency.

* The *copyin* primitive is complicated in the general case by the rather strange semantics of **prober**, which only checks accessibility of the first and last pages in the address range it is given. This forces software to loop over each page involved in the *copyin*. The looping checks are not needed in the code which fetches arguments for system calls, because there are at most 16 bytes of direct arguments to a system call.

process, the way in which the system works is unaffected, and the code is then somewhat easier to understand.

These changes reflect only local optimization of the code; the substance of system call handling has not been changed. It is simply the case that **calls** sequence used by the C compiler for procedure calls is relatively expensive. Expansion of small routines and machine dependent primitives in critical paths is an important technique that can be used to quickly and easily mitigate routine call slowness when profiling or other measurements show this to be necessary.

It should be noted that the use of **calls** and passing of routine parameters on the stack in the current VAX C compiler is different from the way VMS is coded.† VMS makes almost exclusive use of the **jsb** and **rsb** instructions to avoid the overhead of **calls**, and has stylized conventions for assignment of registers across assembly language routines so that pushing and popping of data on the stack can be avoided. This basic mechanism is a good deal faster than **calls** if register usage can be optimized carefully, but tends to make the code harder to change.

Perspective. Our departmental VAX running 20 users in the afternoon does an average of about 100 system calls a second. Under the old system the basic overhead for these 100 system calls was about 3.5% of the available processor time. The faster system call interface reduces this to about 1.4%.

The fact that the primitives for critical sections (the *spl* set priority level routines) were implemented by **calls** to the two instructions implementing them accounted for nearly 1/3 of the time in a *read* or *write* system call. Since over half of all system calls are reads or writes, a simple inline expansion of these primitives accounted for more improvement for reads and writes than the changes to the basic system call routines.

Context switching

The context switching tests attempted to measure how fast the systems could pass control from one process to another. Kashtan measured that VMS could switch between two processes at a net rate of 2000 switches per second using the "event flag" mechanism to signal process exchange. On UNIX, using the *kill* system call as a signalling mechanism, Kashtan found that the maximum switching rate was 210 per second. He estimated that the basic switching rate of the two systems was 5600 switches per second on VMS and 425 per second on UNIX. He concluded:

> "UNIX, as currently implemented, has to do considerably more work when scheduling a process. In addition, UNIX must do a context switch to process number 0 in order to make the decision as to which process to be run next. Even this cannot explain the greater than 10 to 1 difference in the performance of the two systems."

We will see that this difference involves no great mystery, and that, in fact, UNIX can be made to context switch nearly as fast as VMS does simply by changing the (assembly-language) primitive support of context switching to match the hardware. Remaining differences in timings are not the result of "inefficiencies" in UNIX, but from the close fit of the VMS strategy with the VAX architecture.‡

The VAX instruction set caters to a certain regime of context switching. To make its idea of context switching amenable to UNIX, it sufficed to include the C library routines *setjmp* and *longjmp* in the system to handle internal non-local goto's, and to provide a new context switch primitive *resume* that corresponds to a **svpctx** followed by a **ldpctx**.

There were two further problems with the context switching primitives: as on the PDP-11, the per-process system stack and control information were kept in kernel address space. It is much more efficient on the VAX to keep this information in the P0 or P1 region where it will be remapped when **ldpctx** is used. This change was made by moving this information to the base of the stack.

† VMS is written in assembly language.

‡ Specifically, blocked jobs are queued on linked lists with **insque** (insert in queue instruction) for removal with **remque** (remove from queue instruction) while UNIX uses subroutines which hash sleeping jobs. The overhead of calling the hashed *sleep* and *wakeup* routines and searching the hash chains will account for the remaining difference in time.

The final problem was the one Kashtan noted; that the system switched to process 0 each time before switching to the eventual target for the switch. This was done only to allow the system to idle in process 0. This is cleaner than idling on an arbitrary process because, e.g., the process that called *swtch* might have just swapped itself out, and we would then be running on system control information in memory that had been released. There is actually no problem in not switching to process 0, since the system cannot run anything but interrupt code until we come out of the idle loop in *switch*.

After these changes, the times for context switching improved dramatically, dropping by more than a factor of 5 to about 400μsec per switch. Examination of the remaining overhead revealed that there were several small routines being called in critical paths in the *sleep* routine. This was simply eliminated by an inline expansion of the code.

After the above changes had been made, we measured the system context switch time and broke it down as follows:

blocking time in *sleep*	50 μsec
rescheduling time in *swtch*	60μsec
resume primitive	110μsec
unblocking time in *wakeup*	50μsec

Thus the context switching time had been reduced to 270μsec, a factor of seven improvement.

In practice, there is a further efficiency issue here that is not pointed out by the benchmarks. The *swtch* routine used a search over a linked list testing to find the highest priority job on the list. The VMS system uses the **ffs** (find first set bit), **insque** and **remque** instructions and an array of run-queues ordered by priority to make this selection as rapid as possible. We coded this new *swtch* routine (it is about 10 lines of assembly language), and changed the system so that only truly runnable jobs were on the run queue (the old system left runnable jobs on the run queue even when they were swapped out). This allows the *swtch* primitive to run in time independent of the length of the run queue.

Perspective. Let us try to get some perspective for timesharing UNIX systems (such as the machine where this paper is being prepared). The system is currently supporting about 20 users, and doing roughly 50 context switches per second. The original code, which ran in about 2 milliseconds per context switch would have cost 10% of the machine in context switching. A version of the system changed to not switch to process 0 in *swtch* ran in about 800μsec per context switch, resulting in a average context switching overhead of about 4% of the machine. The current system, with all the changes mentioned above, uses roughly 1.3% of the machine in context switching.

If applications need absolutely fastest possible context switching time, then UNIX would have to be changed so that the calls to *sleep* and *wakeup* were less expensive. Roughly half of the 100μsec spent in these routines could be saved by writing them in assembly language and calling them with **jsb** instead of **calls**. They would then not have the 16μsec overhead of **calls** and **ret** and could use registers 0-5 for scratch work instead of saving and restoring registers 6-11, the registers normally used by the C compiler for register variables. Measurements of the incidence of *sleep* and *wakeup* in our environment do not justify such a change.

IPC Mechanisms

The measurements here were of the time for a process to send either 4 or 512 byte packets to another process. The system call overhead and context switch overhead reductions improved performance here, as did the inline expansion of the priority level routines. Finally, a few primitives (file lock and unlock, and user/kernel data movement) were defined as macros or partially expanded inline to save time.

The improvements measured in 4 byte packet transmission are reported in the following table. The three experiments measured the rate at which a process could send 4 bytes of data to itself, to send 4 bytes to another process, and send 4 bytes to another process and receive a 4 byte reply. The measurements give the number of 4 byte packets transferred each second.

Mechanism	To self	One-way	Two-way
VMS Mailboxes	440	297	363
UNIX Before	370	294	281
UNIX After	819	714	600

Perspective. The overhead in using pipes on UNIX was reduced greatly by simply eliminating unnecessary calls of primitives. The ultimate speed of message passing through pipes should be between 600 and 1000 packets per second, when no buffering is taking place.

We see no reason that the *mpx* multiplexed file mechanism cannot be made to operate at speeds greater than that of the pipe mechanism. Most of the overhead in the pipe mechanism is in the block input/output system, which must be called to access and release the file system blocks of a pipe on each *read* or *write* operation. With *mpx*, messages will be buffered in memory, and this expensive indexing can be avoided. By placing the *mpx* buffers in paged memory, it should be simple to provide fast response when activity is heavy without tying up large amounts of core if buffered data accumulates.

Paging

Kashtan measured performance of the paging system on two kinds of paging activity: sequential and random. He also tried varying, in the random case, the standard deviation of successive references. Never did his paging experiments involve any "memory" in the behavior of the processes, and they modified every referenced page. To help cope with jobs which reference or modify large numbers of pages rapidly, we have changed the system to read and write clusters of pages from the paging device. This helps to minimize the incidence of pageins and pageouts. Write clustering alone make nearly a factor of two improvement in the time taking to run most of Kashtan's benchmarks, since he modifies every page. Read clustering improves the performance of jobs that sequentially access virtual memory, and has a good, but less significant impact on other jobs, provided it is used in moderation.*

UNIX attempts to determine the set of pages that have been recently referenced using software simulation of the page reference bits that are not provided by the hardware. While experiments show that this works well on jobs that are typical of our working environment, jobs such as those run by Kashtan are not well observed by this method. A simple circular or random page replacement algorithm is nearly optimal for the experiments that he made. Disabling the gathering and use of page reference information for "memoryless" jobs, and instead relying more on simple circular or random replacement, more akin to the algorithm used by VMS, improves system performance also. We have added an advisory system call that informs the system that the process will not be well observed by the normal reference information gathering algorithm. This is used by LISP during garbage collection. One can easily imagine processes informing the system of strongly sequential or random paging behavior.

The following table gives the relative times for the sequential and random benchmarks on Kashtan's machine and ours. His benchmark used 8192 pages of process virtual space while ours used only 7500. He had 2 megabytes of real memory while we had only 1.75. These numbers are unfortunately not exactly comparable, and we hope to run the experiment on a 2 megabyte machine in the near future.

Experiment	VMS	Old UNIX	New UNIX
Sequential access	4:32	20:16	6:45
Random access	6:00	17:24	10:37

* An excessive amount of prepaging tends to overload the page replacement algorithm by creating an artificially high demand for memory. We can mitigate this somewhat by not validating the pre-paged pages, forcing a "reclaim" to occur quickly if the page is not to be discarded. Observations show, however, that prepaging more than 2048 bytes (2 of the basic 1024 byte pages that UNIX uses) at a time tends to degrade general system performance.

Kashtan also measured a program with random paging behavior (each successive reference was a random distance from the previous), with varying standard deviations of references. We report here the times Kashtan measured for VMS and UNIX when he made the experiment, and also measurements on our system before and after the changes mentioned above were introduced. Kashtan altered the basic paging strategy in VMS for these tests; the strategy used in UNIX here was always the one we use during general timesharing.

Deviation	VMS	SRI UNIX	Old Berk UNIX	New Berk UNIX
1	0:16	0:16	0:23	0:23
10	0:18	0:16	0:24	0:24
30	0:25	0:18	0:66	0:44
40	0:47	1:08		1:34
50	1:21	3:54	4:37	2:38
60	1:53	5:46	6:21	3:13
80	3:04	6:38	8:47	4:53
100	3:27	8:26	10:28	6:12

These measurements are somewhat less than satisfactory for two reasons: the machines had different configurations making the results more difficult to interpret. Also, the experiments were too short, and therefore the start-up time for the system to reach a stable state has not been factored out of the results.†

Perspective. The changes to the paging system to introduce clustering and for special treatment of processes that are not well handled by the normal techniques markedly improve the performance of UNIX on Kashtan's benchmarks. For the jobs we normally see on our system the improvement was not as noticeable. The effect of these changes on a standard synthetic workload that we had previously run to evaluate system performance was negligible.

Of far more importance to system performance was a revision of the swap scheduling algorithm. The previous version of the system used an oldest job first strategy for swapping out runnable jobs. Changing the system to instead swap out the oldest of the *n* largest jobs and then slowing down the rate of swapping had dramatic effects under the heavy loads we have been encountering. This change is of much more importance to the typical UNIX installation than the other changes mentioned above.

Conclusions

In selecting between UNIX and VMS as an operating system for use in the ARPA Image Understanding and VLSI research communities, the UNIX system was chosen primarily out of concern for portability. For our purposes within the Computer Science Department at Berkeley, we have chosen UNIX because of its pliability to meet our needs.

In the short term, there are areas of the UNIX system that are still suffering growing pains from the porting of the system to larger machines. We believe that the simplicity and modularity of the system, which are the keys to its portability, are also the reasons why UNIX is easy to tune, as this paper has demonstrated.

We have by no means made all the changes to the system that will be needed by the ARPA community. The throughput of the UNIX file system is more than adequate for our current timesharing applications, but will not be great enough for large image processing applications, nor will it support page clustering to the file system, which will be essential if large files are to be shared among processes and paged from the file system.

Changes to support such facilities have been designed, and implementation of these facilities is

† For example, the UNIX simulation of software reference bits is disabled when there is a large amount of free memory and a certain amount of time (15 to 20 seconds) is required after free memory drops below a threshold before the reference information begins to become available. This interval is of the same length as the experiments, clouding the results.

not difficult, because of the simplicity of the current system. We believe that we can, with minor changes to the file system structure, increase the throughput sufficiently to support most of these applications.† In the long term, a different basic file system organization may be needed. We are examining other file system schemes, such as extent based file systems, confident that we can easily change the system to incorporate whatever scheme we decide upon. This flexibility, essential for long-term viability of the system, is the reason we choose to use UNIX.

† We plan to initially implement a file system, based on the current *inode* structure, but which maintains a pool of 8192 byte buffers as well as a pool of 1024 byte buffers. By judiciously allocating these 8192 byte contiguous chunks to large files, and allowing programs which need access to large amounts of data to read such large blocks directly into their address space (without copying them in and out of the system buffer cache), we should achieve an significant improvement in file system throughput on applications which (for example) sequentially access large amounts of data. Such a clustering scheme will also allow the paging system to cluster the write-back of pages which are being shared by processes after being "mapped" from files.

LICENSE AGREEMENT

THIS LICENSE AGREEMENT is made and entered into this _____ day of _____, 19___, by and between THE REGENTS OF THE UNIVERSITY OF CALIFORNIA, a California corporation, hereinafter called "LICENSOR", and _____, a _____ having its principal office at _____, hereinafter called "LICENSEE";

WITNESSETH:

WHEREAS, LICENSOR owns and is the proprietor of the copyright of a certain computer program entitled, "Third Berkeley Software Distribution (3BSD)"; and

WHEREAS, LICENSEE desires to obtain from LICENSOR, and LICENSOR desires to grant to LICENSEE, a license to use the aforementioned computer program;

NOW, THEREFORE, in consideration of the mutual covenants, conditions and terms hereinafter set forth, and for other good and valuable consideration, LICENSOR hereby leases to LICENSEE the physical property described on annexed Schedule A ("Licensed Material") subject to a non-transferrable, nonexclusive license ("License"), which is hereby granted to LICENSEE, to use such Licensed Material upon the terms and conditions hereinafter set forth; and LICENSEE hereby accepts such lease subject to the License solely upon such terms and conditions.

1. **Term.** The term of this Agreement shall commence on the date hereof, and, unless sooner terminated as hereinafter set forth, shall extend indefinitely.

2. **Charges.** As a fee for the use of the Licensed Material, LICENSEE shall pay LICENSOR a duplication charge of two hundred dollars ($200.00). LICENSEE may obtain new releases of the Licensed Material as LICENSOR may from time to time make available at a duplication charge of two hundred dollars ($200.00). Such new releases as are purchased by LICENSEE shall by subject to the terms and conditions of this Agreement. Such fee is due and payable when this License Agreement is returned, signed by the LICENSEE, and with a copy of the LICENSEE's UNIX/32V† Agreement.

Such fee does not include local, state or federal taxes, and LICENSEE hereby agrees to pay all such taxes as may be imposed upon LICENSEE or LICENSOR with respect to the ownership, leasing, licensing, rental, sale, purchase, possession or use of the Licensed Material.

3. **Maintenance and Update Services.** Neither maintenance services nor update services are included in this Agreement. As used in the Agreement, the term "maintenance services" includes notice to LICENSEE of latent errors in the Licensed Material and rectification thereof.

4. **Title.** LICENSEE agrees that the Licensed Material is, and shall at all times remain, the property of LICENSOR. LICENSEE shall have no right, title or interest therein or thereto except as expressly set forth in this Agreement. However, those portions of the Licensed Material which are modifications of UNIX/32V and are so indicated on schedule A, are also governed by the LICENSEE's agreement with Western Electric.

5. **Duplication and Disclosure.** LICENSE agrees that all Licensed Material shall be held in confidence, that such Licensed Material is provided for the exclusive use of LICENSEE, on the following CPU, namely _____, Serial No. _____ located at its _____ facility,

† UNIX is a trademark of Bell Laboratories

and any single replacement thereof, provided, that written notice of the replacement and its Serial Number is first given to LICENSOR. The LICENSEE warrants that this machine is licensed, by agreement with Western Electric, for using of the UNIX timesharing system, version 7 (UNIX/32V), dated _____. The Licensed Material 3BSD shall not be duplicated, except as reasonably necessary to LICENSEE's use of the Licensed Material under this Agreement or disclosed to others in whole or in part without the express written permission of LICENSOR. IN PARTICULAR, LICENSEE AGREES THAT THE SOURCE FORM OF LICENSED MATERIAL SHALL NOT BE DISCLOSED TO OTHER LICENSEES WHETHER OR NOT SUCH OTHER LICENSEES HAVE CURRENT VERSIONS OF THE LICENSED MATERIAL. Such prohibitions on disclosure shall not apply to disclosure by LICENSEE to its employees and consultants if and to the extent that such disclosure is reasonably necessary to LICENSEE's use of the Licensed Material and provided that LICENSEE shall take all reasonable steps (including, but not limited to, all steps that LICENSEE takes with respect to information, data, and other tangible and intangible property of its own that it regards as confidential or proprietary) to ensure that such Licensed Material is not disclosed or duplicated in contravention of the provisions of the Agreement by such employees or consultants.

6. Warranty and Limitation of Liability. LICENSOR MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO ANY MATTER WHATSOEVER, INCLUDING WITHOUT LIMITATION, THE CONDITION OF THE LICENSED MATERIAL, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

LICENSOR shall not be liable for, and LICENSEE hereby assumes the risk of and will release and forever discharge LICENSOR, its agents, officers, assistants and employees thereof either in their individual capacities or by reason of their relationship to LICENSOR and its successors in respect to any expense, claim, liability, loss or damage (including any incidental or consequential damage) either direct or indirect, whether incurred, made or suffered by LICENSEE or by third parties, in connection with or in any way arising out of the furnishing, performance or use of the Licensed Material. In any event LICENSOR's liability to LICENSEE on any ground, including but not limited to negligence, shall not exceed a sum equal to the fee paid to LICENSOR by the LICENSEE hereunder except as provided in paragraph 7 hereunder entitled "Patent and Copyright Indemnity".

7. Patent and Copyright Indemnity. LICENSOR will defend the LICENSEE against a claim that a program supplied hereunder infringes a U.S. patent or copyright, LICENSOR will pay the resulting cost and damage awards provided that:

a. The LICENSEE promptly notifies LICENSOR in writing of the claim; and

b. LICENSOR has sole control of the defense and all related settlement negotiations.

If such claim has occurred, or in LICENSOR'S opinion is likely to occur, the LICENSEE agrees to accept noninfringing replacement programs from LICENSOR, if available, or, if not, to return the program on written request by LICENSOR. The LICENSEE will pay only those charges which were payable prior to the date of such return. LICENSOR has no liability for any claim based upon the combination, operation or use of any program supplied hereunder with equipment or data not supplied by LICENSOR, or with any program other than or in addition to the program supplied by LICENSOR if such claim would have been avoided by use of another program whether or not capable of achieving the same results, or based upon modification of any program supplied hereunder.

This indemnity does not cover any material originally supplied to LICENSEE by Western Electric under LICENSEE's UNIX/32V license.

The foregoing states the entire obligation of LICENSOR with respect to infringement of patents and copyrights.

8. Alterations and Modifications. LICENSEE shall make any alterations, variations, modifications, additions or improvements to the Licensed Material, at its own risk and expense

for its own use and merge it into other program material to form an updated work, provided that, upon discontinuance of the License for such Licensed Material the Licensed Material supplied by LICENSOR will be completely removed from the updated work and dealt with under this Agreement as if permission to modify had never been granted. Any portion of the Licensed Material included in an updated work shall be used only on the designated CPU and shall remain subject to all other terms of this agreement.

9. **Inspection.** LICENSOR shall have the right at all reasonable times to inspect the premises of LICENSEE subject to all LICENSEE'S industrial security and other rules then in effect at LICENSEE'S premises; to determine and verify LICENSEE'S compliance with this Agreement.

10. **Default.** If with regard to any of the Licensed Material, LICENSEE fails to pay any charge provided for herein within ten (10) days after written notice that the same is overdue and payable, or if LICENSEE with regard to any item or items of Licensed Material fails to observe, keep or perform any other provisions of the Agreement required to be observed, kept or performed by LICENSEE, LICENSOR shall have the right to exercise any one or more of the following remedies:

(a) To terminate the License herein granted;

(b) To declare the entire amount of any fee payable under Paragraph 2 hereinabove for the entire term of this Agreement immediately due and payable as to any or all items of Licensed Material without notice or demand to LICENSEE;

(c) To sue for and recover all fees then accrued or thereafter accruing, with respect to any items of Licensed Material;

(d) To take possession of any or all items of Licensed Material without demand or notice, wherever they may be located, without court order or other process of law. LICENSEE hereby waives any and all damages occasioned by such taking of possession. No taking of possession shall constitute a termination of this Agreement as to any item of Licensed Material unless LICENSOR expressly so notifies LICENSEE in writing;

(e) To terminate this Agreement as to any or all items of Licensed Material;

(f) In the event of any unauthorized use of the Licensed Material, including, but not limited to, unauthorized disclosure to third persons or use by LICENSEE of the material at facilities other than those identified in Paragraph 5 above, LICENSOR shall at its option have the right in addition to its other remedies, to recover from LICENSEE an amount equal to (i) the sum LICENSOR would have charged the person or persons obtaining the benefit of such unauthorized use of the Licensed Material, plus (ii) any amount received by LICENSEE on account of such unauthorized use;

(g) To have the obligations of LICENSEE hereunder specifically performed and to have any threatened or actual breach by LICENSEE enjoined, it being acknowledged with respect to the obligations of LICENSEE under Paragraph 5 hereof that such equitable relief is the only adequate remedy;

(h) To pursue any other remedy at law or in equity. Notwithstanding any said repossession, or any other action which LICENSOR may take, LICENSEE shall be and remain liable for the full performance of all obligations on his/her part to be performed under this Agreement. All such remedies are cumulative, and may be exercised concurrently or separately.

11. **Legal Expenses.** In case legal action is taken by either party to enforce this Agreement, all costs and expenses, including reasonable attorney's fees, incurred by the prevailing party in exercising any of its rights or remedies hereunder or in enforcing any of the terms, conditions, or provisions hereof shall be paid by the other party.

12. **Assignment.** Without the prior written consent of the other, neither party shall (a) assign, transfer, pledge or hypothecate this Agreement, the Licensed Material or any part thereof or any interest therein or ( ) sublet or lend the Licensed Material or any part thereof, or permit the Licensed Material or any part thereof to be used by anyone except as specifically authorized by Paragraph 5 above. Any consent to any of the foregoing prohibited acts shall apply only in the given instance and shall not be deemed a consent to any subsequent like act nor a consent to any other act. In the event either party consents to any prohibited act hereunder, the other shall, without further request, apprise any third party receiving Licensed Material or the use thereof of the restrictions upon use contained in this Agreement. Subject always to the foregoing, this Agreement shall bind and inure to the benefit of the parties hereto, their successors and assigns.

13. **Severability.** If any part, term or provision of this Agreement shall be held illegal, unenforceable or in conflict with any law of a federal, state or local government having jurisdiction over this Agreement, the validity of the remaining portions or provisions shall not be affected thereby.

14. **Governing Law.** This Agreement shall be construed and enforced according to the laws of California as applied to contracts made and to be performed in California.

15. **Paragraph Headings.** The headings herein are inserted for convenience only and shall not be construed to limit or modify the scope of any provision of this Agreement.

16. **Termination.** Upon termination of the lease herein, all Licensed Materials and copies thereof shall be returned to LICENSOR.

17. **Installations.** Under the terms hereof, LICENSEE is entitled to only one installation of Licensed Materials. Additional installations requested by LICENSEE will be made by LICENSOR under the terms and conditions to be separately negotiated.

18. **Entire Agreement.** This Agreement contains all the agreements, representations, and understandings of the parties hereto and supersedes any previous understandings, commitments or agreements, oral or written.

IN WITNESS WHEREOF, the parties hereto have executed this Agreement as of the day and year first above written.

THE REGENTS OF THE
UNIVERSITY
OF CALIFORNIA

By

_____

(Licensor)

By

_____

(Licensee)

AUUGN

The UNIX Time-Sharing System

T. A. Dolotta

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

UNIX is a trademark for a family of computer operating systems developed at Bell Laboratories. Over 1,100 of these systems, which run on small to large minicomputers and on some large computers, are used within the Bell System for program development, for support of telephone operations, for text processing, for control of laboratory experiments, for computer-aided design, and for general-purpose computing. Another 1,300 UNIX systems have been licensed outside the Bell System.

The main objective of the developers of the UNIX system was to develop a computing environment in which they themselves could comfortably and effectively pursue their own work. The result is an operating system of unusual simplicity, generality, and, above all, intelligibility. A distinctive software style has grown upon this base. UNIX software works smoothly together: elaborate computing tasks are typically composed from loosely-coupled, small parts that possess very simple, general-purpose interfaces, and that are, themselves, very often available as "off-the-shelf" software tools.

This talk is an overview of UNIX. A concise, annotated bibliography of publications that describe UNIX is provided.

SHARE 54
March 6, 1980

BIBLIOGRAPHY

[1] *The Bell System Technical Journal,* Vol. 57, No. 6, Part 2 (July-Aug. 1978).
An issue devoted to UNIX; describes UNIX itself and several applications thereof.

[2] Dolotta, T. A., et al. Programmer's Workbench. *Proc. 2nd International Conf. on Software Engineering,* pp. 164-199 (Oct. 1976).
Six papers that describe various aspects of a version of UNIX that is used primarily for program development and text processing.

[3] Dolotta, T. A., and Mashey, J. R. Using a Command Language as the Primary Programming Tool. In: Beech, D. (ed.), *Proc. of the 2nd Working Conf. on Command Languages,* Amsterdam: North Holland (in press).
Highlights the UNIX command language (which is a full programming language) and some of the ways in which it has been used.

[4] Kernighan, B. W., and Cherry, L. L. A System for Typesetting Mathematics. *Comm. ACM,* Vol. 18, No. 3, pp. 151-156 (Mar. 1975).
A readable description of a very neat UNIX facility.

[5] Kernighan, B. W., and Mashey, J. R. The UNIX Programming Environment. *Software—Practice & Experience,* Vol. 9, No. 1, pp. 1-15 (Jan. 1979).
Explains what's good about UNIX.

[6] Kernighan, B. W., and Plauger, P. J. *Software Tools.* Reading, MA: Addison-Wesley (1976).
A textbook about building good software tools similar to those available in UNIX.

[7] Kernighan, B. W., and Ritchie, D. M. *The C Programming Language.* Englewood Cliffs, NJ: Prentice-Hall (1978).
A book that describes in detail the principal language available in UNIX. Essentially all of UNIX is written in C, which has also been "ported" to a number of different computers.

[8] Levine, J. R., and Morrison, J. P. Data Stream Linkage and the UNIX System. *IBM Systems Journal,* Vol. 18, No. 3, pp. 470-475 (1979).
A discussion of some UNIX features.

[9] Lions, J. Experiences with the UNIX Time-sharing System. *Software—Practice & Experience,* Vol. 9, No. 9, pp. 701-709 (Sep. 1979).
An enjoyable article that tells why they like UNIX in New South Wales.

[10] Miller, R. UNIX—A Portable Operating System? *Operating Systems Review,* Vol. 12, No. 3, pp. 32-37 (July 1978).
Tells how UNIX was "ported" to an Interdata 7/32. This issue of the Operating Systems Review *contains two other UNIX-related papers.*

[11] Ritchie, D. M. The Evolution of the UNIX Time-sharing System. *Proc. of the Symposium on Language Design and Programming Methodology,* Sydney, Australia (Sep. 1979).
Ten years later, one of the creators of UNIX looks back.

[12] Ritchie, D. M., and Thompson, K. The UNIX Time-Sharing System. *Comm. ACM,* Vol. 17, No. 7, pp. 365-375 (July 1974).
The original, prize-winning UNIX paper, written by its two creators. A revised and updated version of this paper appears in the first reference above.

Instruction Backup

Jim McKie

Electrical Engineering
Heriot-Watt University

Dave Rosenthal

Architecture
Edinburgh University

9th Jan 1980

1. Introduction

This note describes the routine "_backup" in the Unix system assembler code (m40.s or m45.s), and the respects in which it is inadequate for the newer small 11s (23, 34, 60). It assumes you have a processor handbook for both the 11/45 and one of the smaller processors, and a copy of John Lion's Commentary on the Unix System. If you don't have a Commentary, then the line numbering in the section describing backup on an 11/40 may need clarification - the entry point to _backup in m40.s (_backup:) is line number 1012.

2. Instruction Backup

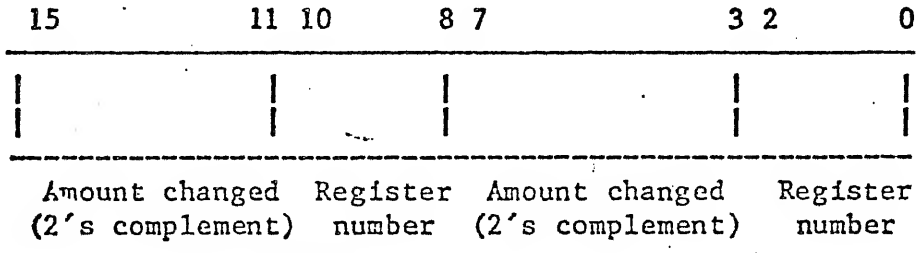
When a user mode segmentation exception occurs, and the user SP is below the stack segment, an attempt is made to grow the stack automatically. The assembler routine "backup" (1012) is used to reconstruct the situation which existed prior to execution of the instruction which caused the trap. "grow" (4136) is used to perform the actual extension. The case where the trap was not due to the SP going below the stack segment is handled by "grow", which returns 0 if the SP is already within the stack segment, and a SIGSEG signal is indicated.

3. Memory Management Status Registers

SSR0 contains abort error flags, memory management enable plus some other information. If any of the abort flags are set, the memory management status registers are

frozen until the flags are cleared, to facilitate error recovery.

The SSR1 register records any autoincrement/decrement of the general purpose registers, including explicit references through the PC. SSR1 is cleared at the beginning of each instruction fetch. Whenever a general purpose register is either autoincremented or autodecremented, the register number and the amount (in 2's complement notation) by which the register was modified is written in SSR1. The information contained in SSR1 is necessary to accomplish an effective recovery from an error resulting in an abort. The low order byte is written first, and it is not possible to for a PDP-11 instruction to autoincrement/decrement more than two general purpose registers per instruction before an "abort causing" reference.



Format of Status Register SSR1

SSR2 is loaded with the 16-bit virtual address at the beginning of each instruction fetch, but is not updated if the instruction fetch fails. Upon an abort, the result of SSR0 bits 15, 14 or 13 being set, SSR2 will freeze until the SSR0 abort flags are cleared.

4. Trap Handling

When a trap occurs, the assembler "trap" routine (0755) stores the contents of the memory management registers (0759) in case they are needed, and the memory management unit is reinitialised.

The process of "backing up" and restarting a partially completed instruction involves:

1. Performing the appropriate memory management tasks to alleviate the cause of the abort;

2. Restoring the general purpose registers indicated in SSR1 to their original contents at the start of the instruction by subtracting the "modify value" specified in SSR1;

3. Restoring the PC to the "abort time" PC by loading it with the contents of SSR2, which contains the value of the virtual PC at the time the "abort generating" instruction was fetched.

The routine backup relies on the ability of the hardware to restart a half executed instruction, e.g. the instruction

```
mov     (r0)+,-(sp)
```

may cause a trap when the attempt is made to push the operand, fetched indirectly through r0, onto the stack. At this point, register r0 has been autoincremented, and must be restored to its original value before the instruction is re-executed after the stack is grown. The memory management available on the larger PDP-11s (44, 45, 55, 70) automatically saves the amount any general purpose register was autoincremented/decremented (a maximum of two registers) in SSR1. However, this register is missing on the smaller processors (23, 34, 40, 60) and the routine "backup" has a lot more work to do, and may fail, as not enough information is saved by the processor. The classic example is on the instruction

```
cmp     -(sp),-(sp)
```

In such a situation, when the source register is the same as the destination register, it is impossible to tell which half of the instruction caused the fault, and no backup is possible.

It will be useful to consider instruction backup when the full set of memory management registers is available (as on the PDP-11/45), before considering the intricacies caused by the absence of SSR1.

5. PDP-11/45 Instruction Backup

Instruction backup with the /45-type segmentation hardware is performed by the following code:-

```
        .globl  _backup
        .globl  _regloc
        _backup:
0001            mov     2(sp),r0
0002            movb    ssr+2,r1
0003            jsr     pc,1f
0004            movb    ssr+3,r1
0005            jsr     pc,1f
0006            movb    _regloc+7,r1
0007            asl     r1
0008            add     r0,r1
```

```
0009            mov     ssr+4,(r1)
0010            clr     r0
0011    2:
0012            rts     pc
0013    1:
0014            mov     r1,-(sp)
0015            asr     (sp)
0016            asr     (sp)
0017            asr     (sp)
0018            bic     $!7,r1
0019            movb    _regloc(r1),r1
0020            asl     r1
0021            add     r0,r1
0022            sub     (sp)+,(r1)
0023            rts     pc
```

Backup is called directly from "trap" (2812) with the address of the saved user registers as argument.

0001 save the pointer to the saved registers in r0; it will be used more than once;

0002 move the lower byte of the saved SSR1 (in "trap") into r1;

0003 lines 13 to 23 are a subroutine which takes the byte specified in r1 and updates the general register specified therein. The lower byte of r1 is of the same format as one of the bytes of SSR1. First, use the next location on the stack as a temporary to hold the "amount changed" (lines 14 to 17); note the implicit sign extension in line 2. Isolate the register number (18), and create a pointer to its stored location (21) using the "regloc" array (2677). Finally, update the stored register value (22) and return;

0004 do the same for the upper byte of SSR1;

0006 reset the PC to that saved in SSR2 at the time of the trap;

0010 indicate no error on return, "backup" cannot fail!

6. Instruction Backup without SSR1

Memory management register SSR1 is missing on the PDP-11/23/34/40/60, and in order to backup the instruction causing a segmentation fault, it must be simulated. The backup procedure is the same as that for an 11/45, but before the registers are updated, the subroutine "backup" is called to simulate the function of the missing register, and put the required offsets and register numbers into the storage space

"ssr+2".

1013 Save the pointer to the saved registers, and the old contents of r2 - it will be used as the missing register SSR1, and call the routine "backup" to simulate the action of the missing register;

1047 initialise "SSR1"; "bflag" is set if we are dealing with a byte instruction, "jflg" is an error condition if it is set;

1050 move the contents of the virtual PC at the time of the abort (SSR2) into r0, and call the routine "fetch" (1222) to get the instruction returned in r0 which caused the abort. Fetch returns -1 on error and also clears the lower byte of r2 if an error occurs - this will be used later.

1053 the top 4 bits of the instruction are used to determine its "type" (a numerical op-code list is very handy). There may not be enough information to determine the type (0 and 10), so a further check is done at line 1066, on the next lower 3 bits. The main categories are double operand, single operand and illegal instruction types (illegal instructions in this case are those such as branch instructions which do not take an operand and could therefore not cause a segmentation exception, as well as true illegal instructions such as 11/45 floating point opcodes). Those of type "illegal" (1188) simply cause jflg to be set and a hasty return to "_backup". If a single operand instruction caused the fault, then the register of its operand is the one which caused the fault, no others being involved, so call "setreg" (1196) to put any register number and amount changed into the lower byte of r2.

Double operand instructions are more complicated; setreg is called on both the source and destination (1115) (note that the source information is put in the high byte, whereas the hardware SSR1 uses the low byte first - it doesn't really matter), and then the source operand is fetched, following through any indirection necessary. This has the effect of clearing the low byte of r2 if a fault occurs (in "fetch"). In this way, if the fault was caused by the source operand, the destination register would not have been altered, and so "backup" should not need to restore it.

1196 The routine "setreg" is interesting as it brings out one or two points about the PDP-11 instruction set. Displacements only occur with register modes 2, 3, 4

and 5 (autoincrement, autoincrement deferred, autodecrement and autodecrement deferred respectively). With modes 3 and 5, the amount changed is always 2 bytes, but with modes 2 and 4, the amount changed can be 1 byte or 2 bytes, depending on whether the instruction is a byte or word instruction. Also, if the register involved is the stack pointer (r6 or SP) or the program counter (r7 or PC), the amount changed is always 2 bytes; this is checked on lines 1206 to 1209.

7. Special Cases

The code in V6's m40.s described by the commentary works correctly on the 11/40. However, the other small machines have certain peculiarities which require special treatment. These include:-

- FP11 instructions. The 11/23+KEF11A, the 11/34+FP11A, the 11/60 and the 11/60+FP11E all process the 11/45 style floating point instructions. The FP11's operands may be 2, 4 or 8 bytes long, and the general registers are adjusted by corresponding amounts.

- The 11/40 always leaves the auto-incremented or decremented registers in their final state, even if a fetch is aborted. In some cases on the other machines, the registers are left in their initial state. So far as is known at present, these cases are:-

  1. FP11 instructions with operands of 4 or 8 bytes length on the 11/60 without FP11E.

  2. Instructions of the form "op (r)+" on the 11/34.

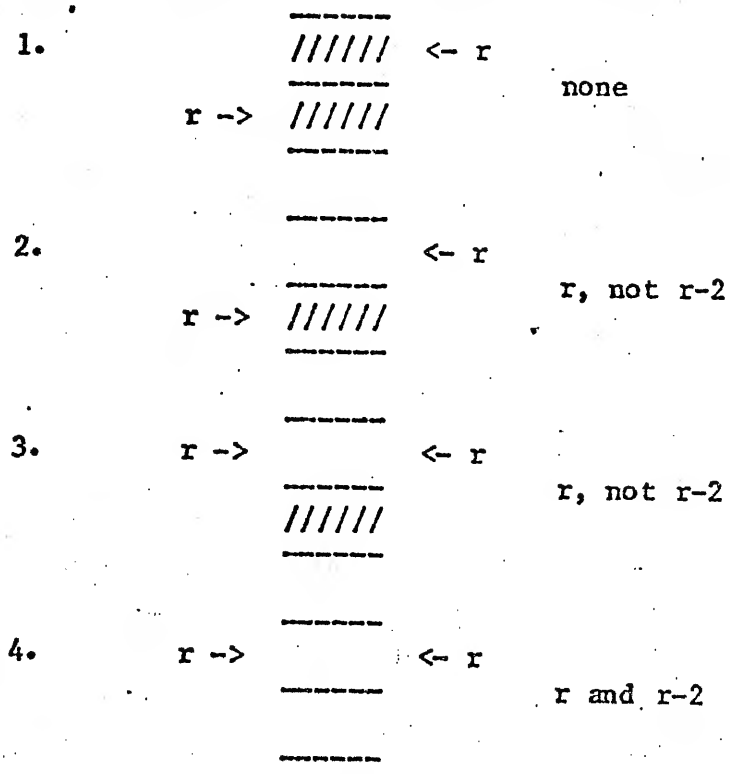
  3. Instructions of the form "op (r)+,dd" on the 11/34, iff the source faults.

  Of course, if the registers are in their initial state, it is unnecessary (and incorrect) to adjust them. However, "grow()" (4136) examines the sp after backup has run, and expands the stack iff the sp points outside the valid stack. If the sp is in its initial state, it may point to a valid stack address, in which case SIGSEG will be incorrectly signalled.

After comparing V6's m40.s, V7's m40.s, Stanford's m34.s and our own m34.s[1], we have written a new version of "backup" which is presently being tested. This supports FP11 instructions, and has facilities for instructions which leave the registers in their initial state. However, it is not a panacea; there is one problem on the 11/34 which

1) Which was distributed with v6+ at Newcastle.

appears insoluble. If the source of an op (r)+,dd faults, there are four possible cases:-



Cases 2 and 3 are indistinguishable after the fact, and so a backup failure must be signalled if this situation is detected. The behaviour of this new backup on the /34 is more conservative than that of Stanford's; it fails rather than backup incorrectly.

PROGRAMMING ANNOUNCEMENT

New Operating System

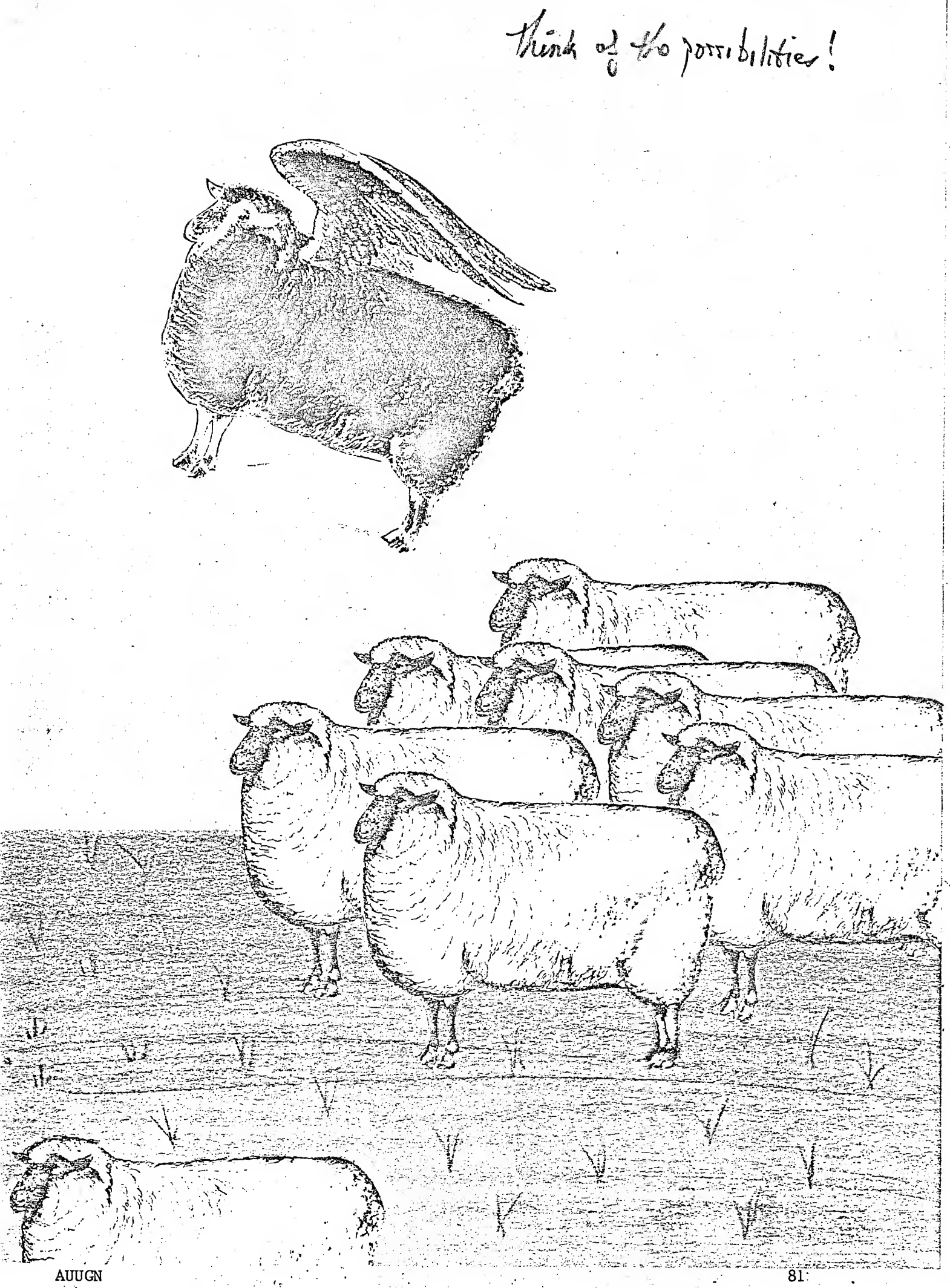
Because so many users have asked for an operating system of even greater capability then VM, IBM announces the Virtual Universe Operating System OS/VU.

Running under OS/VU, the individual user appears to have not merely a machine of his own, but an entire universe of his own on which he can set up and take down his own programs, data sets, systems networks, personnel, and planetary systems. He need only specify the universe he desires, and the OS/VU system generation program (IEHGOD) does the rest. This program will reside in SYS1.GODLIB. The minimum time for this function is 6 days of activity and 1 day of review. In conjunction with OS/VU, all system utilities have been replaced by one program (IEHPROPHET) which will reside in SYS1.MESSIAH. This program has no parameters or control cards as it knows what you want to do when it is executed.

Naturally, the user must have attained a certain degree of sophistication in the data processing field if an efficient utilisation of OS/VU is to be achieved. Frequent calls to non-resident galaxies, for instance, can lead to unexpected delays in the execution of a job. Although IBM, through its wholly-owned subsidiary, the United States, is working on a program to upgrade the speed of light and thus reduce the overhead of extraterrestrial and metadimensional paging, users must be careful for the present to stay within the laws of physics. IBM must charge an additional fee for violations.

OS/VU will run on any IBM x0xx equipped with Extended WARP Feature. Rental is twenty million dollars per cpu/nanosecond.

Users should be aware that IBM plans to migrate all existing systems and hardware to OS/VU as soon as our engineers effect one output that is (conceptually) error-free. This will give us a base to develop an even more powerful operating system, target date 2001, designated "Virtual Reality". OS/VR is planned to enable the user to migrate to a totally unreal universe. To aid the user in identifying the difference between "Virtual Reality" and "Real Reality", a file containing a linear arrangement of multisensory total records of successive moments of now will be established. It's name will be SYS1.est.


think of the possibilities!

Unix Installation Equipment List

organisation _____
department _____
address _____
city _____
region _____
telephone(organisation) _____
telephone(computer room) _____
telephone(dial up line) _____

correspondent _____ extension _____
system manager _____ extension _____

licence(commer/academ) _____ cpu type _____
Unix versions licenced _____ cpu num _____
Unix versions run _____ memory (Kb) _____
Other systems run _____

	manuf	model	agent	qty	description
disks					
tapes					
terminal					
printers					
plotters					
displays					
other					

networks _____
_____
_____

cassette	dectape	floppy-disk	paper-tape	card-reader	
modem	auto-dial	a/d converters	floating point	cache	

I object to this information being made available to:
Unix group members _____
Non group members _____

Unix Installation Software List

organisation _____
department _____
favoured media _____

utility	version/description	status

Unix Luug Installation List

Luug name _____

chairman _____
address _____
_____
phone _____

correspondent _____
address _____
_____
phone _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____

organisation _____
department _____
representatives _____